Departamento de Ingeniería de Sistemas y Automática

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO FIN DE GRADO

RED DE SENSORES Y ACTUADORES INALÁMBRICOS DE BAJO COSTE PARA UNA VIVIENDA

Autor: Guillermo Cañada Valverde

Tutor: Mohamed Abderrahim Fichouche

Director: Alberto Brunete González

Leganés, Marzo de 2013

Título: RED DE SENSORES Y ACTUADORES INALÁMBRICOS DE BAJO COSTE PARA UNA VIVIENDA

Autor: Guillermo Cañada Valverde

Tutor: Mohamed Abderrahim Fichouche

Director: Alberto Brunete González

EL TRIBUNAL

Presidente: _____

Vocal:

_____

Secretario:

_____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

A toda la gente en general que me ha apoyado con esto, desde el principio de la carrera hace ya 5 años y que nunca ha perdido la fe en mí durante el camino.

En especial a mi familia, a mi novia y a Alberto Brunete, director del proyecto.

A mi padre por cuyas ideas y conocimientos me han aportado mucho a lo largo del proyecto.

A mi madre por la insistencia mostrada para centrarme, que no siempre es fácil.

A mi novia por seguir día a día el proceso y apoyarme en los momentos en los que parecía que no lo conseguiría.

Y a Alberto Brunete por haberme apoyado en esto desde el primer momento a pesar de todas las adversidades y de toda su ocupación. Han sido muchas horas de problemas y más problemas pero siempre con plena disposición y ganas de ayudarme, ¡Gracias Alberto!.

Por supuesto agradecer también a Mohamed Abderrahim por tutorizarme el proyecto haciéndolo posible y por sus ideas y recomendaciones.

RESUMEN

Este proyecto es un estudio sobre la automatización inalámbrica de una casa con una red de sensores y actuadores (WSN) Zolertia Z1 mediante 6LoWPAN.

El concepto principal del proyecto es la "domótica fácil", el objetivo siempre ha sido conseguir domotizar una casa con sensores y actuadores de plug-and-play, de un modo fácil e intuitivo para cualquier persona.

La tecnología inalámbrica utilizada, 6LoWPAN, es un estándar que posibilita el uso de IPv6 sobre redes basadas en el estándar IEEE 802.15.4. Su bajo consumo hace que su gasto energético se limite a un par de pilas AA de 1,5 voltios cada varios años.

El proyecto en sí consta de dos partes: la parte principal, el 90% del proyecto, es el desarrollo con la tecnología inalámbrica, la creación de la estructura y de las comunicaciones entre los diferentes sensores y actuadores de la red con el Gateway y de este con la interfaz de control en el PC.

Al estar la tecnología aun en progreso, y ser los dispositivos Zolertia Z1 para desarrollo (no aún para comercialización), los programas que se han creado son programas de desarrollo que abarcan todas y cada una de las posibilidades, aunque en la realidad los dispositivos tendrían programas más pequeños que solo abarcasen algunas de las funcionalidades desarrolladas.

La otra pequeña parte del proyecto es una visualización, simulada en parte, de cómo quedaría el programa final que controlaría el usuario, con dispositivos con diferentes programas y funcionalidades concretas.

Todo ello ha sido programado usando el SO Contiki, que hace de enlace entre el ordenador y las redes de sensores.

Palabras clave: WSN, domótica, inalámbrica, 6LoWPAN, Zolertia Z1.

ABSTRACT

This project is a study about the wireless automation of a house with a Zolertia Z1 sensors and actuator network (WSN) through 6LoWPAN.

The main concept of the project is the “easy domotic”, the target has always been getting to domotize a house with plug-and-play sensors and actuators, in an easy and intuitive way to everybody.

The wireless technology used, 6LoWPAN, is a standard that allows the use of IPv6 on networks based on standard IEEE 802.15.4. Its low consumption makes its energy use to be limited in a pair of AA batteries each some year.

The project himself consists in two parts: the main part, the 90% of the project, is the wireless technology development, the creation of the structure and the communications between the different sensors and actuators of the network with the Gateway and between the Gateway with the control interface at the PC.

As this technology is still in progress, and as the Z1 devices are being developed (not yet for commercialization) , the programs that has been created are programs for development that cover every possibilities, although in the reality the devices would have smaller programs which cover just some of the developed functionalities.

The smaller part of the project is the visualization, partly simulated, of how the final program that the user would use looks, with devices with different programs and concrete functionalities.

All of this has been programmed using the Contiki SO, which work as a link between the computer and the sensor network.

Keywords: WSN, domotic, wireless, 6LoWPAN, Zolertia Z1, Contiki.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1 DESCRIPCIÓN DEL PROYECTO

En los últimos años la domótica ha tenido una gran evolución, siempre acompañada por los avances tecnológicos en el campo de la informática, de la electrónica y de la automatización.

Todos sabemos los beneficios que tiene la domótica para la gente en general, haciendo la vida más fácil y automatizando tareas en una casa, y también la ayuda que da a gente con capacidades limitadas, como ancianos o discapacitados, con tareas domésticas habituales, como puede ser subir una persiana o poner la calefacción sin peligro.

A pesar de todos los avances, cuando uno piensa en domótica o en casa inteligente le viene a la cabeza o una casa nueva que ya incorpore las funcionalidades, o tener que realizar una obra, por pequeña que sea, para poder implantar un sistema domótico en una casa. Esto supone no solo incomodidades sino disponer de una cantidad de dinero que no tiene mucha gente.

Para poner la domótica a disposición del gran público se necesita superar esto, y con la aparición de nuevas tecnologías inalámbricas de bajo coste y consumo estamos cada vez más cerca. Este proyecto trata precisamente de eso, de ese acercamiento a la "domótica fácil", a una domótica de plug-and-play de bajo coste.

Con el protocolo inalámbrico 6LoWPAN se consigue la primera parte, será el estándar utilizado en este proyecto. Este protocolo tiene un consumo bajo de energía que permite mantener muchos años los dispositivos sin necesidad de cambiar las baterías. Además, el SO(Sistema Operativo) Contiki tiene una gran cantidad de bibliotecas para diferentes dispositivos (entre ellas el Zolertia Z1, el dispositivo utilizado en este proyecto) entre las que se incluyen funciones con comunicaciones inalámbricas que facilitan mucho la labor.

La segunda parte, el conseguir la "domótica fácil", se logra mediante la programación adecuada de la red de sensores y actuadores y su relación con la interfaz de usuario.

Así, se obtendrá por una parte con una red de sensores y actuadores, que en el desarrollo del proyecto han sido programados con todas las funciones posibles y finalmente se han realizado pequeños programas mono función que simulasen sensores y actuadores reales, como pueda ser un detector de luz, un interruptor, un detector de temperatura o un pequeño motor, todos ellos con su correspondiente antena para la comunicación inalámbrica.

Estos dispositivos harán una solicitud para entrar en la red inalámbrica, gestionada por el Gateway.

Este Gateway, también un dispositivo Zolertia Z1, centralizará las comunicaciones de la red y las transmitirá al interfaz de usuario.

Todos los dispositivos tendrán una IPv6 asociada con la que se comunicarán entre ellos.

Finalmente se desarrollará la interfaz de usuario programada en QT Creator desde donde se podrá configurar y visualizar la red creada y modificarla a conveniencia.

1.2 OBJETIVOS

El objetivo final del proyecto es la creación de la red de sensores y actuadores y su control desde el PC. Para conseguir este objetivo se han marcado varios objetivos secundarios intermedios:

- Búsqueda de información básica acerca de las motas Zolertia Z1 y del sistema operativo Contiki. Para ello se han usado foros de desarrolladores, tutoriales y guías de usuario.

- Comunicación inalámbrica unicast entre dos dispositivos, ambos mandando y recibiendo mensajes. Después de conocer los diferentes tipos de comunicaciones entre motas (broadcast, unicast..) se ha trabajado hasta conseguir la comunicación correcta entre dos dispositivos, basándose en los programas básicos unicast-sender y unicast-receiver de las librerías de contiki.

- Comunicación vía Puerto USB entre el dispositivo Gateway (en adelante, Gateway, a secas) y la interfaz de usuario programada en QT Creator. Para ello se han buscado ejemplos de comunicaciones mediante el puerto serie para adaptarse a la interfaz de usuario del proyecto. En un inicio se ha utilizado PuTTY para el desarrollo de dichas comunicaciones por el puerto USB.

- Comunicación bidireccional entre la interfaz de usuario y un dispositivo inalámbrico. Para ello se han buscado ejemplos no solo de emisión de comunicaciones por puerto serie sino de recepción y almacenamiento en buffer.

- Creación del protocolo de comunicaciones y de la estructura de todo el conjunto. Para ello se han mirado diferentes tipos de eventos resultantes de las comunicaciones vía Puerto Serie y vía comunicación inalámbrica que ayudasen al programa a "despertar en los momentos en los que se recibiese uno de los anteriores mensajes.

- Adaptación a la interfaz de usuario. Una vez establecida toda la comunicación de ida y vuelta con dispositivos inalámbricos, el objetivo ha sido la adaptación de todos los comandos y peticiones de las motas a la interfaz de usuario, para poder realizar comunicaciones correctamente entre Gateway y PC.

1.3 MEDIOS UTILIZADOS

Para la realización de este proyecto se han empleado los siguientes medios y materiales:

- Un PC con sistema operativo Linux-Ubuntu, o en su defecto una máquina virtual del mismo con VMware Network Adapter.
- OS Contiki.
- Software Qt Creator.
- Software PuTTY.
- Software Dia.
- Microsoft Office Starter.
- 3 Dispositivos Zolertia Z1 Platform
- 3 Cables USB: uno para la conexión de un dispositivo Zolertia Z1 a modo de Gateway, y otro de apoyo para leer el output de alguno de los dispositivos inalámbricos (exclusivamente para desarrollo).
- Un sensor de luminosidad ZIG002 Light Sensor.
- 6 baterías para los dispositivos Zolertia Z1 Platform.

1.4 ESTRUCTURA DE LA MEMORIA

La memoria está dividida en nueve apartados en los que se desarrolla de forma diferenciada las partes del proyecto en un orden en el que se pueda seguir fácilmente el progreso del mismo. A continuación se hace un pequeño resumen de cada apartado:

Capítulo 1. Introducción y Objetivos.

Es una visión general del conjunto del proyecto, así como de las metas que se persiguen. Además, se indica de forma breve los materiales utilizados.

Capítulo 2. Estado del Arte.

Se introduce todo lo relacionado con el estado actual del mercado en el campo de las redes de sensores inalámbricas y los protocolos de comunicación existentes, así como de los diferentes dispositivos que soportan este tipo de comunicaciones de bajo consumo.

Incluye una exposición de las soluciones que ofrece el mercado en estos campos.

Finalmente, trata brevemente las posibilidades entorno a los software de desarrollo C++ actuales.

Capítulo 3. Estructura de la Red.

En este capítulo se explican los requisitos previos e instalaciones que se tienen que llevar a cabo en el sistema para poder programar la red de sensores

(Instalación de Linux, Contiki…etc).

También se detalla la arquitectura del conjunto de la red, ya con una solución elegida de las propuestas en el anterior apartado.

En este caso se desarrollan 6LoWPAN, Zolertia Z1 y QtCreator, a fin de ver que ofrecen y sus características hardware y software.

Capítulo 4. Diseño y desarrollo de las motas.

Durante este capítulo se concentra la mayor parte de información acerca de la programación de la red.

Se detalla la programación tanto de los dispositivos inalámbricos como del Gateway, analizando las comunicaciones entre dispositivos.

Incluye el programa de desarrollo y varios programas uni-función, simulando como sería un dispositivo en la realidad.

Capítulo 5. Interfaz de control (QT Creator)

Este capítulo se centra en la interfaz de control desde donde el usuario podrá interactuar con la red de dispositivos, tanto desde su programación en si como su comunicación vía puerto USB con el Gateway.

Capítulo 6. Instalación y puesta en marcha.

Durante este capítulo se explica cómo implantar la red inalámbrica una vez programada. Se detalla el proceso de inclusión de motas en la red y como asociarlas entre ellas.

También se explica cómo proceder ante un reinicio de Gateway o de alguno de los dispositivos inalámbricos del sistema.

Capítulo 7. Gestión de proyecto

Se realiza un desglose de los costes del proyecto, tanto materiales como de personal, así como una exposición de la planificación que se ha llevado en el desarrollo del proyecto mediante un diagrama de Gantt.

Capítulo 8. Conclusiones y trabajos futuros

Finalmente, en este capítulo se analizan posibles mejoras sobre el proyecto y áreas donde podría ampliarse, así como las utilidades que podría tener en un futuro.

CAPITULO 2

ESTADO DEL ARTE

En este capítulo se muestra el estado actual de las soluciones que ofrece el mercado en los ámbitos de los protocolos inalámbricos, de los dispositivos que soportan dichos protocolos, es decir, en general, del Wireless Networking.

Finalmente también se incluye una breve exposición de la actualidad en torno a las interfaces de desarrollo para PC.

2.1 PROTOCOLOS INALÁMBRICOS

Las redes de sensores inalámbricas (WSN, Wireless Networking) están formadas por un número indeterminado de dispositivos (llamados comúnmente motas) que se comunican entre ellos informaciones provenientes de sus sensores o trasmitiendo órdenes a sus actuadores.

En este proyecto, una de estas motas centralizará la información, el Gateway.

Para la comunicación de este tipo de redes se utilizan protocolos de comunicación inalámbrica que creen redes privadas y que tengan un bajo consumo energético.

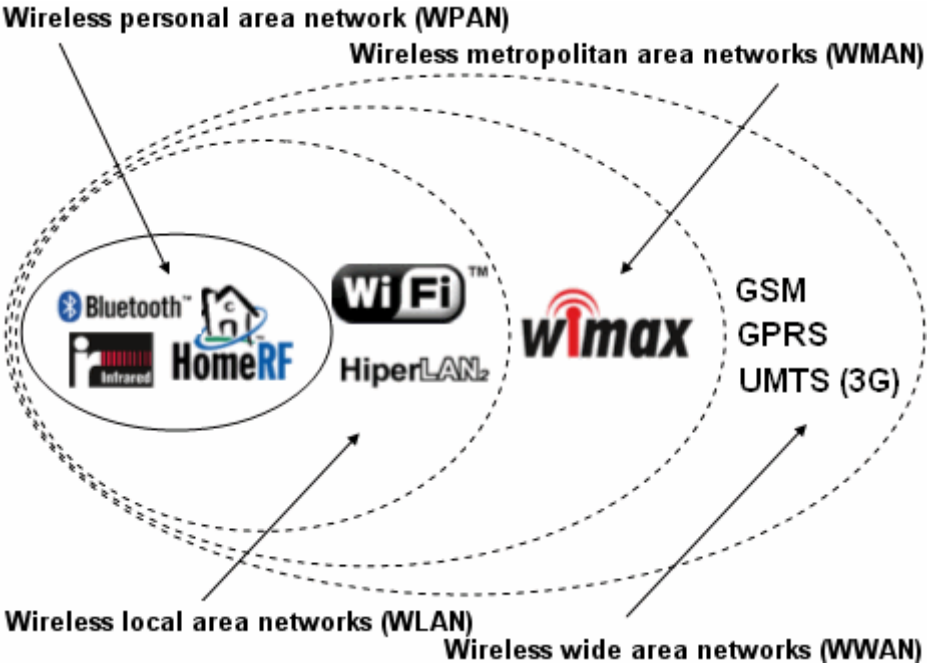


Figura 1[]. Al ser una red doméstica, se va al círculo más pequeño, las WPAN.

-[] Figura 1. Greenlinx (2014). Wireless. Consultada el 12 de febrero de 2014 en: http://www.greenlinx.us/solutions/services/wireless-images-wpan-wlan-wman-wwan

Dentro de todo el conjunto de la comunicación inalámbrica, el estudio se va a centrar en lo que se necesita para el proyecto: las WPAN (Wireless Personal Area Network), ya que se necesita crear una red personal y privada.

Estas redes normalmente tienen un alcance de unos pocos metros.

Para elegir dentro de la variedad de soluciones que se ofrecen dentro del espectro WPAN (Estándar IEEE, Institute of Electrical and Electronics Engineers 802.15), primero hay que tener en cuenta las necesidades que se tienen:

-Complejidad.

-Precio.

-Consumo energético (relacionado invariablemente con la tasa de datos y la distancia de alcance).

-OS y dispositivos de soporte.

	ZIGBEE Y 802.15.4	GSM/GPRS CDMA	802.11	Bluetooth
Aplicación principal	Monitorización Y Control	Amplias áreas de voz y datos	Internet de alta velocidad	Conectividad dispositivos
Vida de la batería	Años	1 semana	5 dias	1 semana
Ancho de banda	250 Kbps	+128 Kbps	+11.000 Kbps	720 Kbps
Rango alcance	+100 metros	Varios kilometros	1-100 metros	10+ metros
Ventajas	Bajo consumo	Alcance, calidad	Velocidad, ubicación	Coste

Tabla 1[]. Comparativa entre los diferentes tipos de tecnologías inalámbricas.

- [] Tabla 1. Webdelcire (2012). Comenzando con Zigbee. Consultada el 12 de febrero de 2014 en: http://webdelcire.com/wordpress/archives/1714

A continuación se explican las diferentes soluciones dentro del protocolo 802.15:

BLUETOOTH (IEEE 802.15.1) [1]

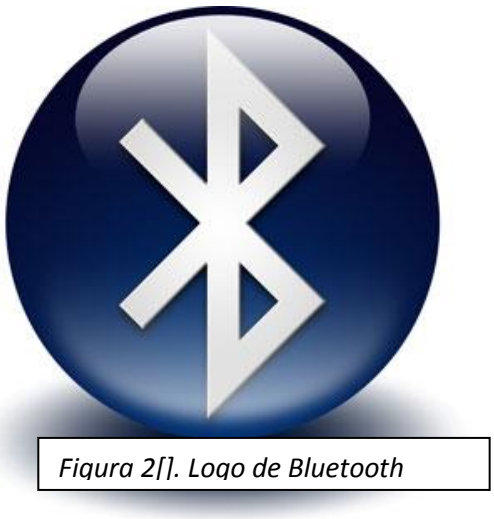


Figura 2[]. Logo de Bluetooth

Bluetooth fue diseñada para sustituir al cable en las transmisiones de información entre dispositivos móviles. Realiza la transmisión de datos mediante radiofrecuencia.

Además, permite crear pequeñas redes inalámbricas.

Este protocolo se usa sobre todo en dispositivos electrónicos personales como teléfonos móviles, tablets u ordenadores portátiles.

El problema que existe respecto al blueetooth se encuentra en su consumo eléctrico, ya que es mayor que el que se necesita en este proyecto.

Además, este consumo aumenta cuanto mayor es el alcance que se necesite, así, existen bluetooth de diferentes clases:

Clase	**Potencia máxima permitida**	**Potencia máxima permitida**	**Alcance**
	(mW)	(dBm)	**(aproximado)**
Clase 1	100 mW	20 dBm	~30 metros
Clase 2	2.5 mW	4 dBm	~10-5 metros
Clase 3	1 mW	0 dBm	~1 metro

Tabla 2[]. La potencia (y el consumo) aumenta en paralelo con el alcance.

- [] Figura 2. Latinpost(2013). GPS garmin nuvi. .Consultada el 12 de febrero de 2014 en: http://latinpost.mx/114440/gps-garmin-nuvi-2797lmt.html

- [] Tabla 2. Wikipedia (última edición 2014). Bluetooth. Consultada el 12 de febrero de 2014 en: http://es.wikipedia.org/wiki/Bluetooth

IEEE 802.15.2 [2]

Este estándar estudia las coexistencias entre redes IEEE 802.15 con otros dispositivos inalámbricos que usen otros estándares, como Wifi (802.11) u otros.

IEEE 802.15.3 [2]

Este estándar también es llamado el WPAN de alta velocidad. Obviamente esa alta velocidad implica un mayor consumo. Está pensado para su aplicación en transferencias de elementos multimedia, teniendo una tasa de transmisión de datos muy elevada.

IEEE 802.15.4 [2]

Por último se llega al estándar 802.15.4, también llamado de baja velocidad.

Este estándar está creado para dispositivos con una tasa de transmisión de datos muy baja pero con un consumo mínimo, en general de muy baja complejidad.

Dentro de este estándar existen ciertas variantes, como ZigBee o 6LoWPAN.

ZIGBEE [3]

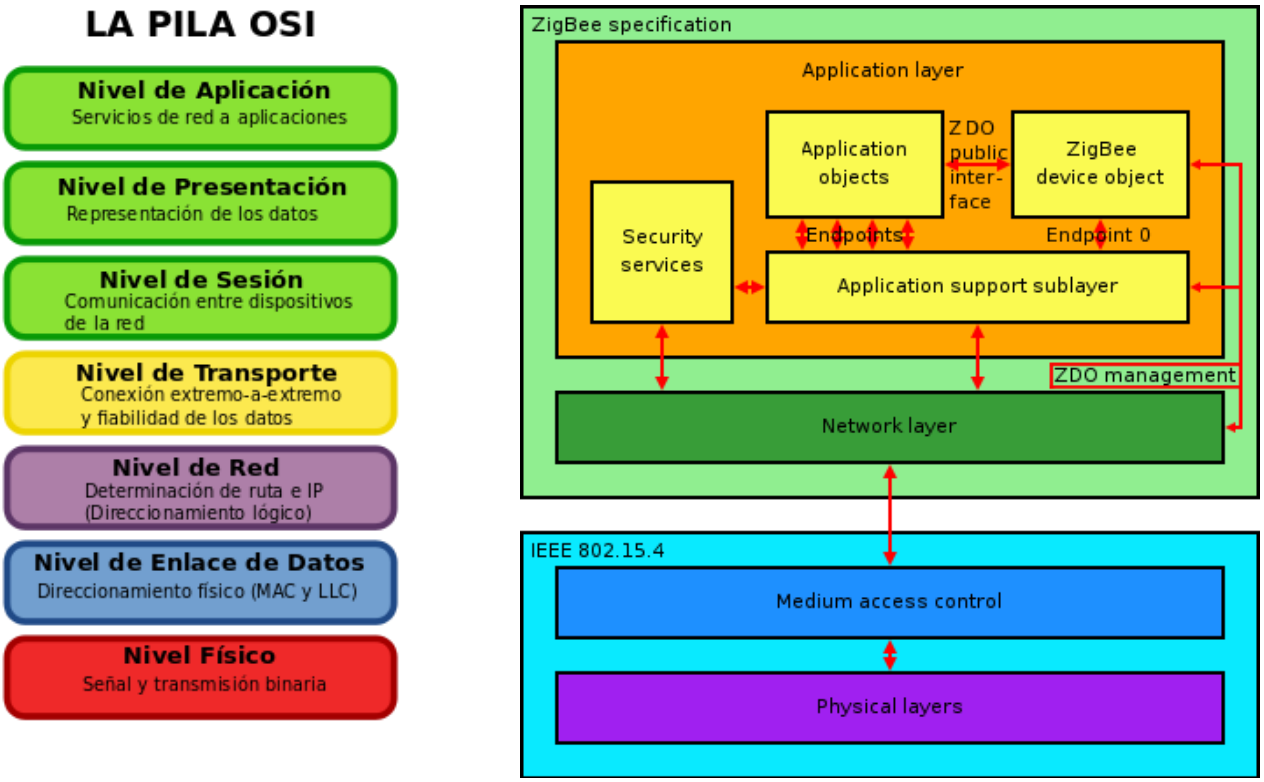
ZigBee es un conjunto de protocolos de alto nivel dentro de las comunicaciones inalámbricas que está basado en el estándar 802.15.4.

Su objetivo son los dispositivos con baja tasa de transferencia de datos y máxima vida útil de baterías.

Figura 3[]. Logo de ZigBee Alliance

- *[] Figura 3. Vesternet(*). Zigbee. Consultada el 12 de febrero de 2014 en: http://www.vesternet.com/zigbee*

Al respetar el estándar IEEE 802.15.4 este delimita los dos niveles más bajos de las transmisiones, es decir, tanto el nivel físico como el nivel de enlace de datos (MAC, Media Access Control, y LLC, Logical Link Control) vienen dados por el protocolo.



Figuras 4[] y 5[] .Por un lado la Pila OSI de comunicaciones, por otro lado la especificación de la misma para ZigBee.

Las redes con ZigBee se desarrollan en diferentes topologías, ya pueden ser de malla, de árbol o de estrella (en función de su disposición). Todas ellas tienen en común los tipos de dispositivos:

Dispositivos coordinadores (Gateways). Controlan la red y los caminos que deben seguir las transmisiones. El mismo tipo de dispositivo puede funcionar de coordinador o de router.

- *[] Figura 4. Wikipedia (última edición 2014). Pila OSI. Consultada el 12 de febrero de 2014 en: http://es.wikipedia.org/wiki/Modelo_OSI*

- *[] Figura 5. Wikipedia (última edición 2013). Zigbee (especificación). Consultada el 12 de febrero de 2014 en: http://es.wikipedia.org/wiki/ZigBee_(especificaci%C3%B3n)*

Dispositivos periféricos, dentro de los cuales hay routers o dispositivos finales:

Dispositivos con todas las funciones (Routers). Interconecta dispositivos separados de la red y ofrece un nivel de aplicación para código de usuario.

Dispositivos finales. Solo pueden hablar con su nodo padre. No tienen la capacidad de ser coordinadores o routers.

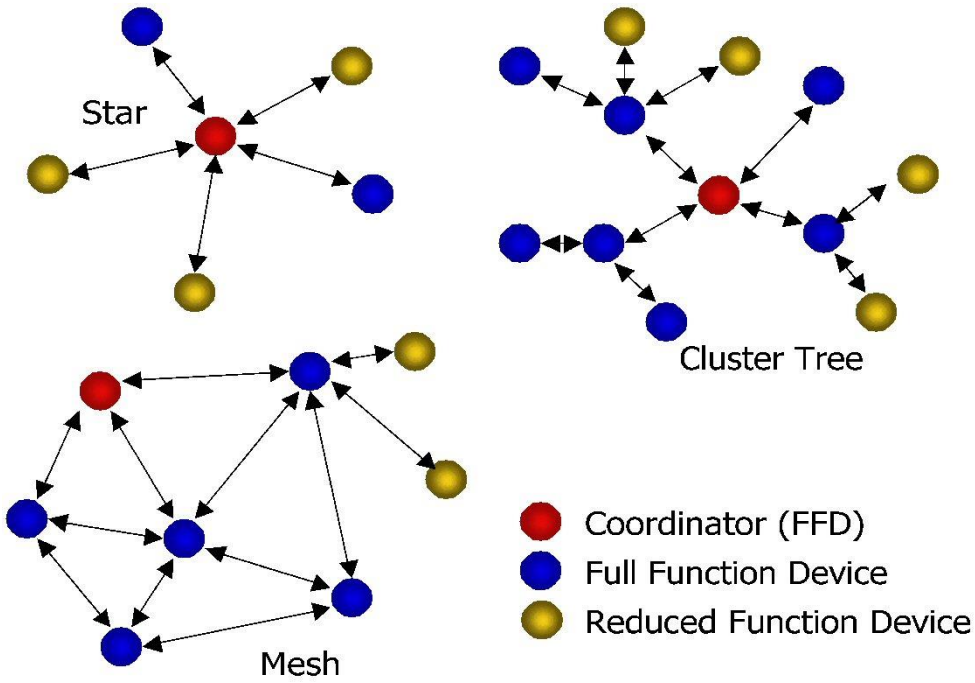


Figura 6[]. Tipos de redes en ZigBee, con los diferentes tipos de dispositivos.

Resaltar que ZigBee Alliance es un grupo de empresas que fomentan el uso de este conjunto de protocolos y por ello cuenta con un impulso interesante de cara a lograr ser un estándar para este tipo de comunicaciones como Wifi lo es para el internet de alta velocidad inalámbrico.

-[] Figura 6. EEtimes(2005). ZigBee Socs provide cost-effective solution. Consultada el 12 de febrero de 2014 en: http://www.eetimes.com/document.asp?doc_id=1273396

6LOWPAN [4]

6LoWPAN es un acrónimo de IPv6 over Low power Wireless Personal Area Networks.

Esto es, es un estándar que posibilita el uso de IPv6 sobre el IEEE 802.15.4, y ahí reside su gran diferencia con ZigBee, como se puede ver en la figura siguiente:

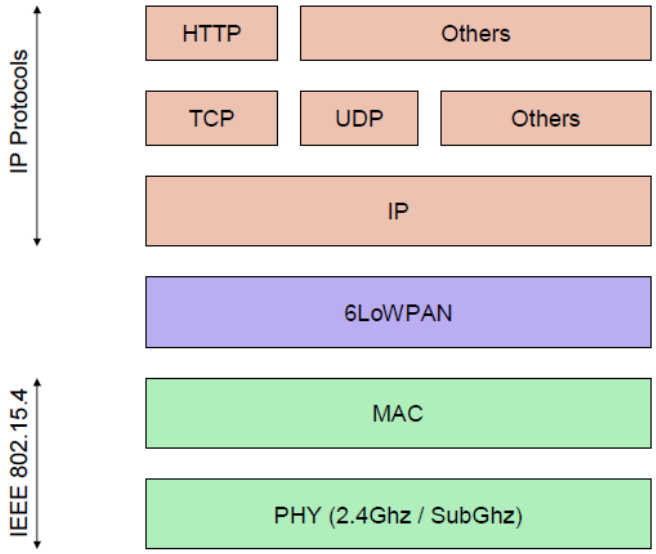


Figure 1 - IP Layers

Figura 7[]. La pila OSI especificada para 6LoWPAN.

Como se ha visto antes, el estándar ZigBee delimita casi todos los niveles más altos (hasta el nivel de aplicación) de protocolo, mientras que 6LoWPAN tan solo afecta al nivel de red.

A efectos prácticos esto significa que 6LoWPAN da una libertad mayor en los niveles altos de comunicación, a lo que sumándole que 6LoWPAN es totalmente libre y desligado de empresas, tiene como consecuencia la falta de uniformidad.

- *[]Figura 7. Perytons(2012). Challenges in analyzing 6LoWPAN/ZigBee IP Networks. Consultada el 12 de febrero de 2014 en: http://www.perytons.com/zigbee/whitepapers/page/3/*

En la siguiente figura se puede ver con mayor claridad cómo se realiza el paso desde el nivel físico y la capa MAC hasta tener paquetes de ipv6 vía 6LoWPAN:

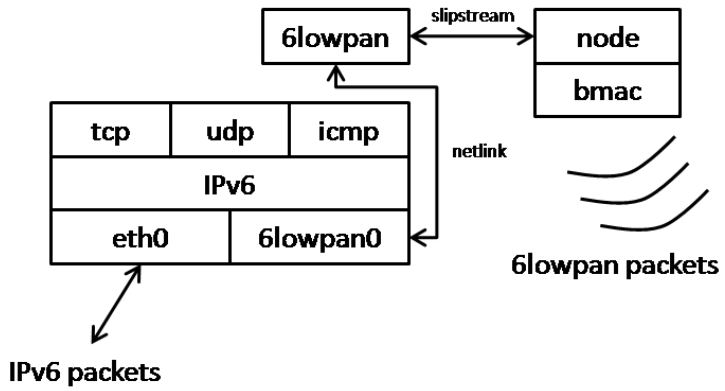


Figura 8[]. De la capa física a IPv6.

Las razones que suelen venir asociadas a escoger 6LoWPAN sobre ZigBee son las siguientes:

-Utilizar IPv6 proporciona conectividad con otras redes IP.

-El número de direcciones (enorme) que utilizar una IPv6 implica.

-Se puede utilizar con diferentes sistemas operativos de open source como Contiki, TinyOS o MansOS.

El debate entre el uso de ZigBee o 6LoWPAN es frecuente entre los programadores de este tipo de dispositivos.

- [] Figura 8. Nanork (2011). 6LoWPAN. Consultada el 12 de febrero de 2014 en: http://www.nanork.org/projects/nanork/wiki/6LoWPAN

2.2 DISPOSITIVOS (MOTAS)

El mercado de las motas o nodos inalámbricos ha crecido muchísimo en los últimos años, y tratar de exponer todas las soluciones posibles sería imposible.

El estudio va a centrarse en cinco tipos de dispositivos, para así poder ver las similitudes y diferencias que existen entre diversos los tipos de motas.

TELOSB [5]

Crossbow ofrece una gran variedad de modelos tanto de motas como de sensores y otros productos que pueden interesar a la hora de realizar una red de sensores.

Centrándose en las motas, todas ellas basadas en el estándar IEEE 802.15.4, tienen un rango de precios bastante pequeño y tienen muchas similitudes entre ellas.

Por lo general las características son las siguientes:

- IEEE 802.15.4 WSN mota totalmente compatible con la plataforma TelosB.
- Micro controlador TI MSP430F1611 y Chip CC2420 RF.
- Compatible con TinyOS 2.x y ContikiOS.
- Temperatura, humedad relativa y sensor de luz.
- Botones de reset y de usuario.
- 3xLeds.
- Interfaz USB.

En algunos modelos la antena es externa.

Figura 9[5]. Modelo CM5000 con antena interna.

WASPMOTE [6]

Al igual que Crossbow, Libelium, la empresa desarrolladora de Waspmote ofrece multitud de soluciones para el WSN.

La mota de Waspmote carece de antena pero se le pueden incorporar de forma externa no solo antenas para el estándar IEEE 802.15.4 o ZigBee, sino incluso Bluetooth, 3G, GPRS y varios protocolos más.

Figura 10[6]. Mota Waspmote básica.

Figura 11[6]. Antenas para ZigBee y 6LoWPAN respectivamente.

Libelium posee una IDE (Entorno de desarrollo integrado) propio para la programación de sus motas Waspmote.

Waspmote tiene características muy similares a TelosB en cuanto a memorias e interfaces. Tiene algo menos de RAM (8 KB por 10 KB de TelosB) aunque casi el triple de Flash (128 KB por 48 KB de TelosB).

MICAz[5]

Crossbow es la empresa desarrolladora de MICAz (también produce TelosB, entre otros).

MICAz puede trabajar con 2.4GHz y 868/916 MHz.

Al igual que todos los dispositivos anteriores es compatible con IEEE 802.15.4/ZigBee, a 2.4 GHz.

Figura 12[5]. Mota MICAz con antena externa.

Las características de Flash son similares a Waspmote (128K) y al igual que las dos motas anteriores, su rango de alcance de antena alcanza los 100 metros en espacios abiertos y entre 20 a 30 metros en espacios cerrados.

MICAz tiene la posibilidad de conectar expansiones de sensores de luz, temperatura, presión, etc ..

Las motas son programables con los interfaces de control MIB510 o MIB520 que también sirven para visualización de datos

Cualquier mota de MICAz puede funcionar como Gateway estando conectada al ordenador o a la interfaz de control.

ARDUINO [7]

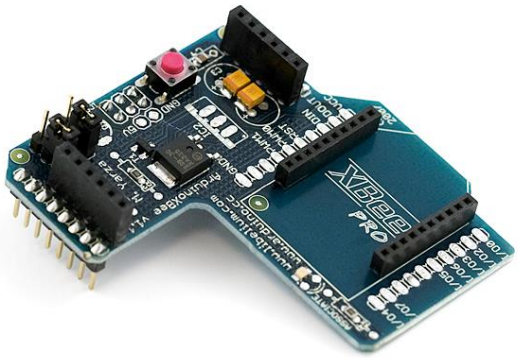
Últimamente la placa Arduino se ha convertido casi en un estándar para la domótica.

Arduino tiene múltiples posibilidades que van desde aplicaciones realmente sencillas, gracias a su bajísimo precio, hasta aplicaciones más complejas que usualmente requieren el uso de "Shields", es decir, de añadidos hardware a la placa básica.

Figura 13[7]*. Placa Arduino UNO sin añadidos*

Su lenguaje propio y sencillo y su interfaz gráfica propia (Arduino IDE) de fácil uso han hecho que hoy en día sea el dispositivo de este tipo más conocido y extendido.

Si bien es cierto que en el tema de creación de redes inalámbricas, el tener que adquirir no solo la placa Arduino en sí sino también una Xbee Shield y una antena externa, hacen que sea más complejo usarlo en este tipo de redes.



Figuras 14[7] *y 15*[7]. *XBee Shield sin antena y antena XBee. Montando los tres elementos tendríamos un Arduino con capacidad de comunicación inalámbrica.*

Aunque con todo esto siguiese siendo la opción más económica, también es la que menos capacidades nos ofrece, teniendo 32 KB de Flash y 2 KB de ram.

ZOLERTIA Z1 [8]

Zolertia fabrica el dispositivo z1, que es uno de los que mejores prestaciones tiene del mercado, aunque también los más caros.

El dispositivo Z1 tiene integrada una antena con capacidad para comunicaciones con IEEE 802.15.4 y ZigBee. Además, esta mota está soportada por varios de los sistemas operativos de software libre más usados con TinyOS y Contiki.

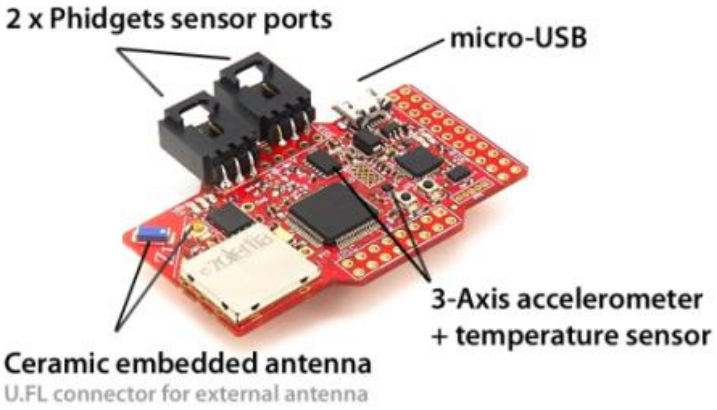


Figura 16[8]. Dispositivo Zolertia Z1.

El dispositivo dispone de un sensor de temperatura, un acelerómetro y un sensor de batería integrado, así como puertos para conectar más sensores externos.

Zolertia ofrece soluciones también para sensores, aunque aún no para actuadores.

Las características técnicas son de la zona media-alta, teniendo 92 KB de Flash y 8 KB de RAM.

En la actualidad no existe ningún software específico para la programación de Zolertia Z1, si bien la propia empresa ha anunciado que está trabajando para tener un IDE propio. De momento puede usarse como banco de pruebas la aplicación Cooja, de Contiki, que sirve como simulador no solo de Zolertia Z1 sino de varios dispositivos más.

2.3 INTERFAZ DE DESARROLLO

Para facilitar la creación del interfaz de usuario, programado en C++, existen muchas soluciones gratuitas en el mercado.

C++ es uno de los lenguajes más habituales en la actualidad y por ello no es difícil encontrar software que facilite al programador el desarrollo de programas en este lenguaje.

En este apartado no se va a especificar cada aplicación en concreto puesto que todas ofrecen más o menos lo mismo con pequeñas diferencias.

Lo que se necesita para el proyecto es una aplicación multiplataforma (es decir, que permita programar en Linux) que facilite la programación en C++. También es necesario que tenga las librerías que se necesiten, incluida SerialPort.h para la comunicación serie, o que puedan incorporarse dichas bibliotecas externamente.

Ultimate++, Zinjai, Adjunta DevStudio, Geany o Sun Studio son, entre otros muchos, programas que cumplen todos estos requisitos.

Algunos de ellos como Ultimate++ cuentan con tecnología BLITZ para compilar más rápidamente C++. La mayoría también cuenta con debuggers.

Para hacer esta interfaz más sencilla y fácilmente manejable también se ha valorado que el software elegido pudiese ser programado mediante Widgets, pasa así darle un componente visual y sencillez al programa.

CodeLite, Code Blocks o QtCreator son ejemplos de programas que permiten usar Widgets.

CAPITULO 3

ESTRUCTURA DE LA RED

En este capítulo se explican todos los pre-requisitos previos a la programación de la red de sensores, desde la adecuación de Linux hasta la puesta en marcha de las motas.

A continuación se detallará la arquitectura de la red, basándose en las soluciones elegidas en cuanto a protocolos de comunicación inalámbricos, motas y software de desarrollo.

3.1 DESCRIPCIÓN DE LA RED

El esquema que se propone es muy sencillo, una red formada por varios dispositivos inalámbricos (motas) y un coordinador y controlador central (PC)(Figura 17):

-El PC, con la interfaz de usuario desarrollada en Qt Creator, crea la red y permite al usuario configurarla.

-Motas: A su vez se dividen entre dispositivos finales (end-devices) y Gateway. EL Gateway (dispositivo Zolertia Z1) transmite las órdenes de la interfaz de control a las motas inalámbricas. Las motas inalámbricas (dispositivos Zolertia Z1) hacen las veces de actuadores o de sensores, según su función.

Todas estas comunicaciones se realizan mediante 6LoWPAN.

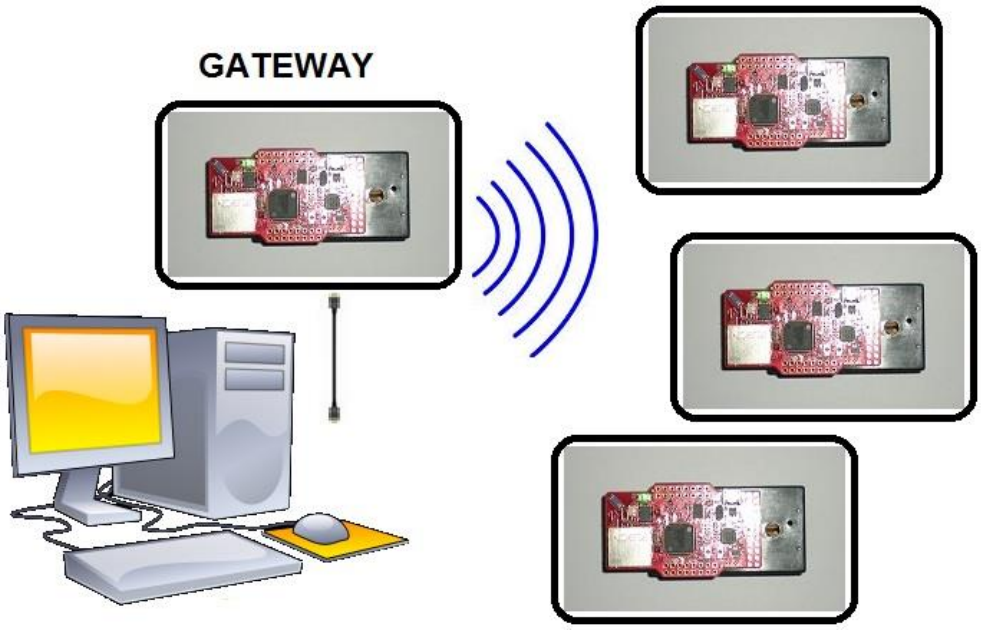


Figura 17. Esquema de la Red. El número de dispositivos inalámbricos es ampliable.

El PC se comunica con el Gateway vía puerto serie, conectado con un cable USB al micro USB de la mota que hace las funciones de Gateway.

Esta comunicación es mediante el ttyUSBX (habitualmente 0, configurable mediante la interfaz de control del usuario) y el UART0 de la mota.

Las motas se comunican a su vez, entre ellas, mediante protocolo 6LoWPAN-IP-UDP (IP a nivel de red, UDP a nivel de transporte), con transmisiones Unicast, es decir, comunicaciones cerradas entre dos motas con un solo emisor y un solo destinatario.

3.2 DESCRIPCIÓN DEL HARDWARE DE LAS MOTAS[8]

Para la función de motas inalámbricas, al igual que para la de Gateway, se ha usado dispositivos Z1 de la marca Zolertia.

Como se ha explicado en la introducción, gracias a las prestaciones del Z1, las motas pueden ejercer las funciones de Gateway y de dispositivos periféricos según su programación.

Zolertia ha desarrollado estos dispositivos Z1 gracias a los avances tecnológicos en diversos campos (computación, sensores, comunicación inalámbrica…) y con este producto está en disposición de interactuar con objetos desde interfaces de control virtuales.

La Z1 tiene alta compatibilidad con otros dispositivos del sector y soporte de OS como Contiki o TinyOS. Además, es capaz de comunicarse siguiendo los protocolos principales de 802.15.4, ZigBee y 6LoWPAN.

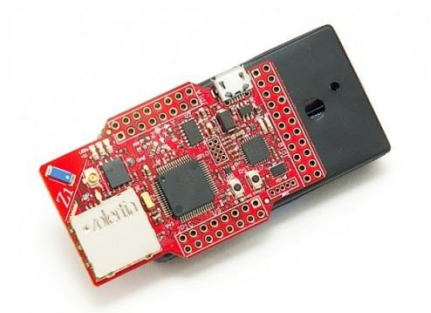


Figura 18[8]. Vista superior del Z1.

El dispositivo puede llevar Phidgets para la conexión de sensores externos (ver Figura 16) o no llevarlos y tan solo hacer uso de los que tiene internamente (temperatura, batería…).

Las aplicaciones del Z1 y sus características son las siguientes:

APLICACIONES:

- Conectividad con Internet of Things (IoT)
- Monitorización de funciones corporales.
- Monitorización ambiental.
- Detectores de emergencia.
- Dispositivos de seguridad y rescate.
- Monitorización de procesos no atendidos.
- Medidas de energía.
- Monitorización de procesos de agricultura.

CARACTERÍSTICAS DEL PRODUCTO:

- Plataforma ideal para el desarrollo de prototipos de WSN.
- Grado de temperaturas ideal para procesos industriales (-40ºC-85ºC).
- Conector de expansión de 52 pins.
- Segunda generación MSP430™, 16-bit MCU 16MHz de muy bajo consumo.
- 2.4GHz IEEE 802.15.4, compilado 6LowPAN y preparado para ZigBee™.
- Acelerómetro digital.
- Sensor de temperatura de bajo consume con precisión de ±0.5ºC (en rango de –25°C ~ 85°C).
- Antena externa opcional vía conector U.FL.
- Conector micro-usb para alimentación y debuggeo.

La comunicación con el Z1 puede hacerse por varias vías. En la siguiente figura se pueden ver los conectores, las posibilidades de comunicación y algunas de las capacidades técnicas del dispositivo.

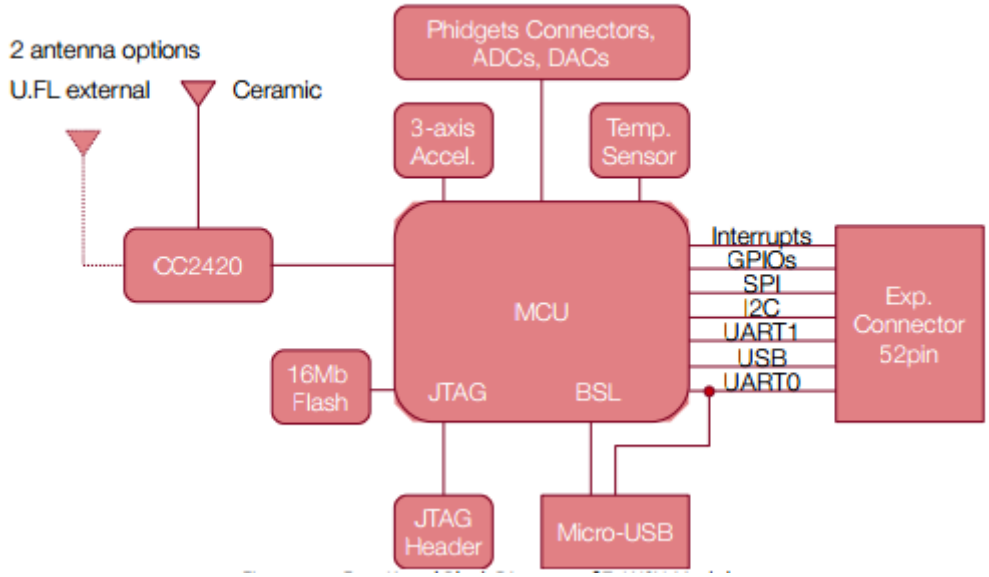


Figura 19[8]. Mapa funcional de bloques Zolertia Z1.

Toda la información sobre la mota, incluida la anteriormente expuesta, puede encontrarse online en:

http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf

Para el uso de las motas inalámbricas también he utilizado uno de los sensores que Zolertia ofrece. El sensor de luminosidad ZIG002 [9] Light Sensor suele usarse en redes inalámbricas de bajo consumo, debido a su poco consumo energético.

El sensor recoge la luz recibida y ofrece un número, que puede variar entre 1 y 40000, en función de la luminosidad.

Technical data	
Supply voltage	2.4V to 3V
Current consumption (vdd=3V at 25°C)	Sleep Typ. 3.2µA Measuring Typ. 0.24mA
Lux range	0.1 to 40000
I2C frequncy bus	Max. 400KHz
Operating temperature range	-30 to +70°C
Board dimensions	35.7x26.5x8.6mm

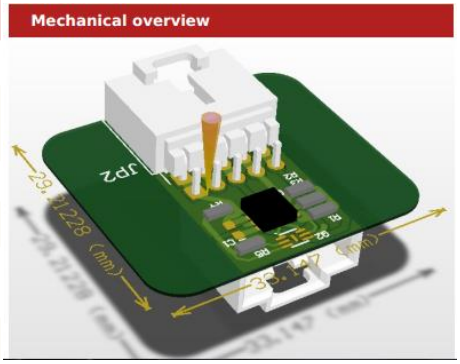


Figura 20[9]. Características técnicas y mecánicas del ZIG002.

Toda la información sobre el sensor, incluida la anteriormente expuesta, puede encontrarse online en:

http://zolertia.sourceforge.net/wiki/images/3/39/Brochure_Zig002.pdf

3.3 FUNCIONALIDAD DE LAS MOTAS

Como se ha dicho en el apartado anterior, todas las motas del sistema tienen el mismo hardware, pero no así la misma programación interna.

Siguiendo una estructura peculiar, que toma características de las formas de malla, de estrella y de arbol, cada mota tiene sus funciones.

Tiene similitud con la estructura de malla porque prácticamente una mota puede comunicarse con cualquier otra.

Tiene similitud con la estructura de árbol porque todas las comunicaciones están centralizadas en el Gateway.

Y finalmente tiene similitud con la estructura de estrella porque los el centro es el Gateway, con el cual se comunican principalmente (aunque no únicamente) los actuadores y estos con los sensores.

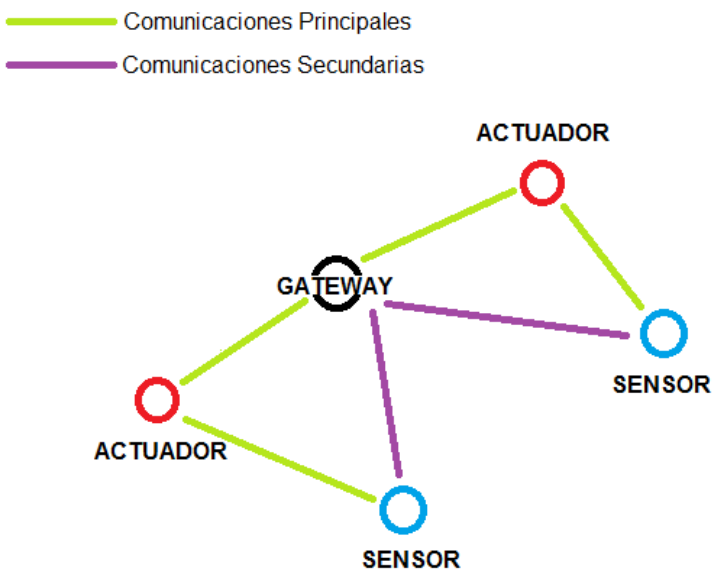


Figura 21. Funcionalidad de las motas en la red.

Las comunicaciones principales entre Gateway-Actuadores implican una disposición contante a la comunicación y para el uso de las diferentes acciones, para la configuración de los mismos.

Las comunicaciones secundarias que se realizan entre Gateway-Sensores tienen como fundamente el incluir a estos últimos en la red y ponerlos en contacto con un actuador al que a partir de ese momento transmitirán la información que capten del medio, siendo este tipo de comunicación Sensor-Actuador también principal.

En realidad el Gateway podrá seguir comunicándose en cualquier momento con los sensores a fin de saber su estado de batería y relación con los actuadores.

Para la simulación de estos sensores y actuadores, además de la funcionalidad de las motas de desarrollo (tanto Gateway como periféricos) con todas las funciones habilitadas, se han creado los siguientes programas mono función:

-Sensor Luminosidad.

-Sensor Temperatura.

-Interruptor.

-Actuador luminoso (LEDS).

En la práctica todos estos intercambios de información tienen una fiabilidad bastante mediocre. Es muy continua la pérdida de algunos paquetes entre las motas cuando la comunicación es inalámbrica (no así entre el Gateway y el PC).

Si bien normalmente no hay problemas de comunicación, o muy escasos cuando acaba de iniciarse la red, es muy común la aparición de problemas cuando los dispositivos comienzan a hibernar.

Debido a que las funciones de comunicación inalámbrica tardan cierto tiempo desde que la mota ha sido iniciada en estar disponibles (no mucho, en torno a 20-30 segundos) se da a menudo que cuando ya ha hibernado una mota, tenga que esperarse de nuevo este tiempo para volver a establecer comunicación eficaz. Algunas veces incluso esa comunicación nunca vuelve y debe reiniciarse la mota inalámbrica para volver a coger conexión con el Gateway.

Algo que también da algunos problemas es cuando varias motas se comunican con una misma mota. Con esto quiero decir, si una vez se programa una asociación entre un

sensor y un actuador no se vuelve a interferir esa comunicación, ésta funciona sin problemas.

Pero si cada cierto tiempo consultamos a uno u a otro corremos el riesgo de que pierdan comunicación entre ambos de forma esporádica o total.

Son muy comunes estas pérdidas de comunicación temporal entre motas, y que , sin aparente cambio, vuelva a funcionar (fuera de paquetes perdidos e hibernaciones).

3.4 SOFTWARE DE LA INTERFAZ DE CONTROL [10]

Para la programación de la interfaz de usuario se ha utilizado el programa Qt Creator.

Qt creator es una aplicación multiplataforma de ayuda al desarrollo de programas en C/C++. Esta IDE fue creada por Trolltech, actualmente Qt Developement Frameworks, para de desarrollo de aplicaciones con librerías Qt.

Qt Creator incluye debugger y un diseñador propio de Widgets.

Además, en Linux, hace uso del compilador de C++ que proporciona la GNU Compiler Collection.

Figura 22[10]. Página de inicio de QtCreator, versión Windows.

El uso de los Widgets en Qt es muy sencillo, debido a su interfaz gráfica y a la posibilidad de añadir recursos externos como imágenes para utilizarlos dentro del programa como fondos de pantalla, botones, etc…

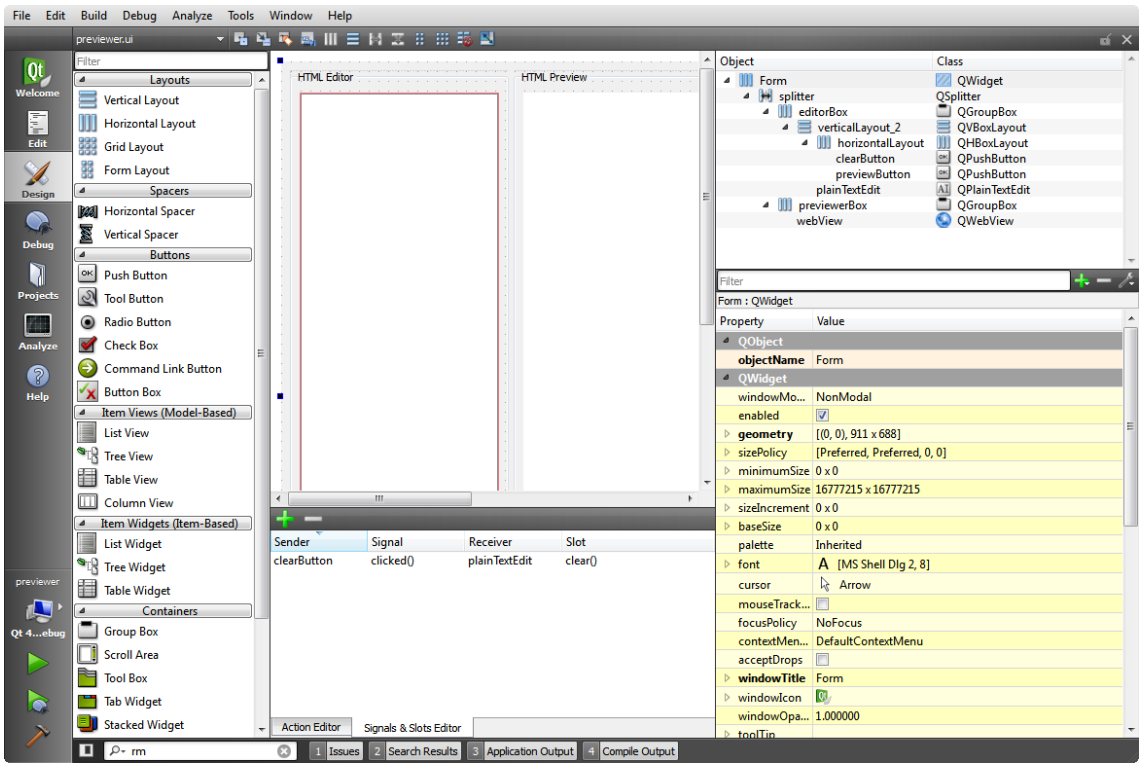


Figura 23[10]. Página de diseño de la Widget, versión Linux.

Se podrá diseñar una Widget contando con una gran variedad de elementos predefinidos que el sistema pone a nuestra disposición, desde los típicos botones hasta diferentes layouts o contenedores para exponer datos.

El programado de elementos se realiza de una forma sencilla, enlazándolo desde esta misma página (figura 23), y programando en C++ habitual (con librerías extra para funciones particulares) lo que interese en cada momento.

3.5 HERRAMIENTAS DE DESARROLLO [11]

Hasta lograr la creación total de la red con el Z1 es útil utilizar el simulador de Contiki, Cooja, para avanzar en el proyecto.

Más adelante, para la programación **real** del Gateway se ha utilizado el programa de editor de textos básico de Linux, posteriormente pasándolo por un compilador y cargándolo en la mota.

Así pues, el proceso de programación de las motas ha sido el siguiente:

1/Editor de textos con programa en C-> compilador en Cooja-> simulación.

Y posteriormente:

2/Editor de textos con programa en C -> compilador en terminal -> subir programa a la mota.

Cooja es un simulador muy completo y con muy buena interfaz gráfica, que permite retocar los programas al momento y te da una maniobrabilidad muy buena para debuggear. Este simulador está incluido en Contiki y es muy sencillo de lanzar:

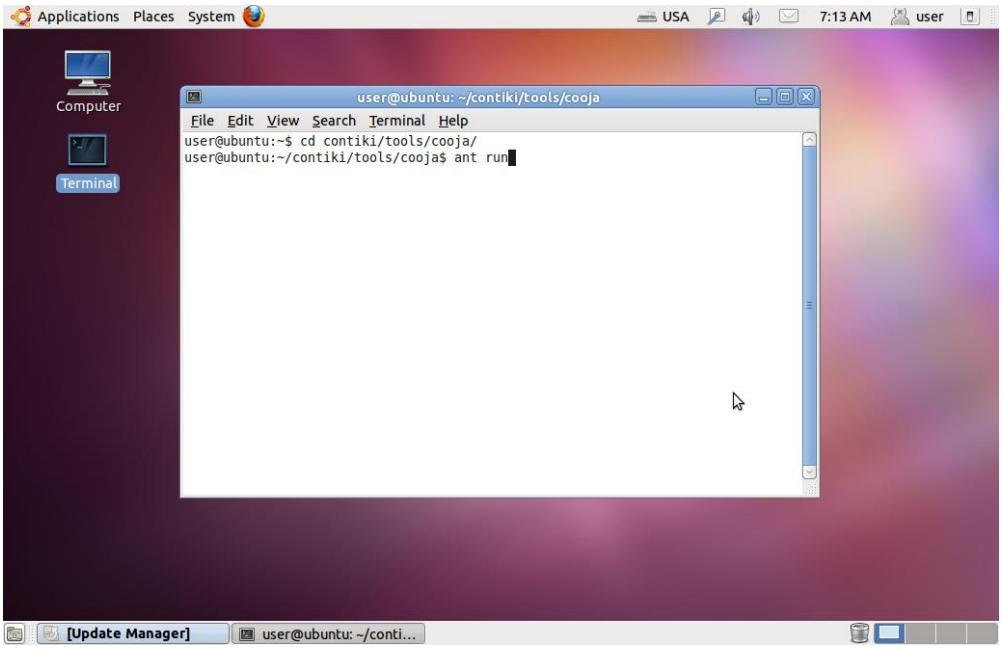


Figura 24[11]. Lanzando Cooja desde el terminal.

El único PERO es la comunicación vía puerto serie que no funciona, y es algo esencial en este proyecto, por lo que solo se ha podido usar Cooja en la parte inicial del mismo.

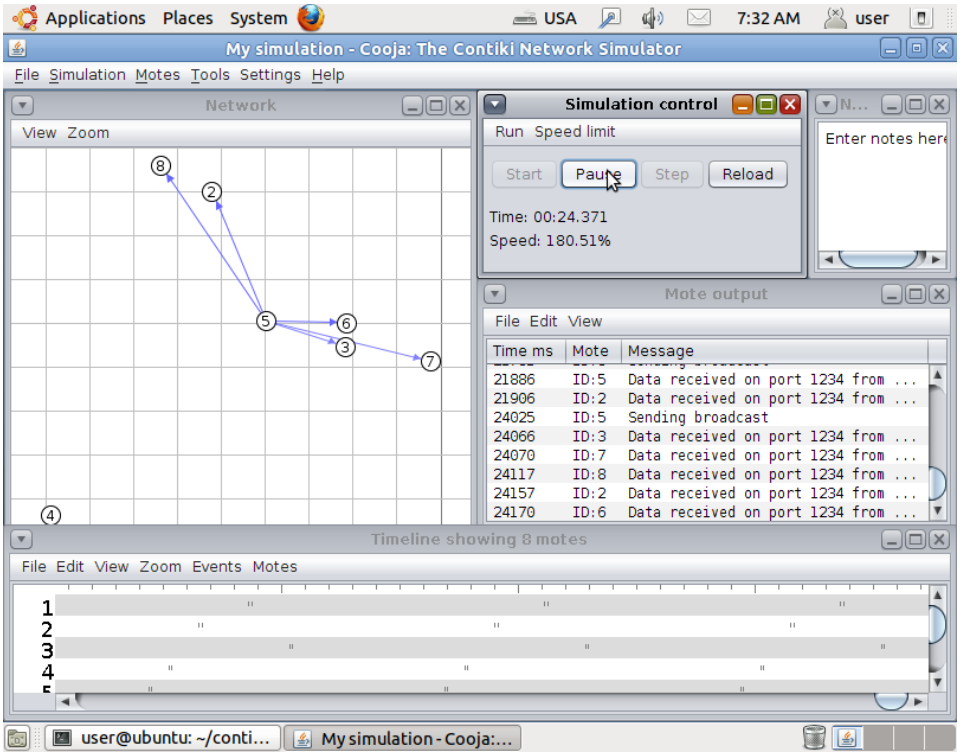


Figura 25[11]. Interfaz gráfica del simulador Cooja.

Cooja nos permite incluir varias motas (en la imagen representadas por círculos con números) con diferentes programas (e incluso diferentes tipos de motas) que cargamos desde el exterior, representando así una localización física de las mismas y la interacción entre motas con diferentes programas.

También permite ver las comunicaciones de radio que se producen entre las motas, sus LEDS, pulsar el botón USR y varias funcionalidades más para tratar de asemejar la simulación a las motas reales en la medida de lo posible.

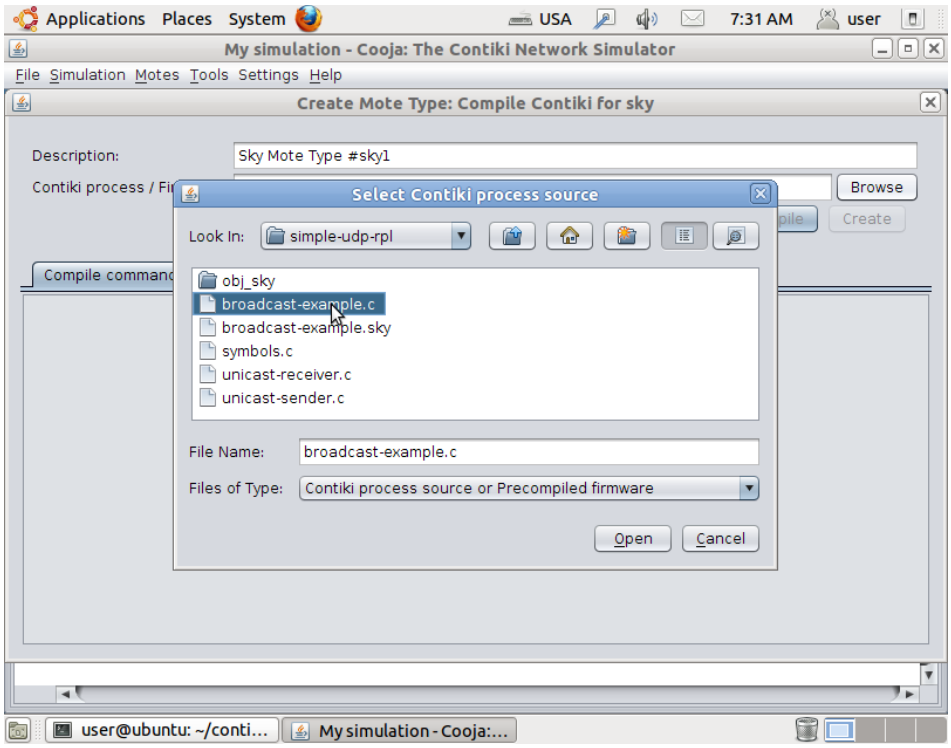


Figura 26[11]. Pantalla de elección de programa, que es posteriormente compilado.

Además, al realizarse incluso la compilación dentro del propio Cooja es muy cómodo para hacer pequeñas variaciones en los programas en C con el editor de textos y con un solo clic, dándole a RELOAD (figura 25), tener de nuevo tu sistema simulado tal y como lo tenías antes pero con los cambios que hayas hecho fuera del Cooja.

Dada la falta de comunicación por puerto serie en Cooja, se hace indispensable usar las motas reales para la realización de las pruebas.

Esto ralentiza mucho el proceso de prueba-error-corrección ya que hay que compilar los programas desde terminal para posteriormente subirlos a la mota:

```
user@instant-contiki:~$ cd contiki-2.6/tools/z1
user@instant-contiki:~/contiki-2.6/tools/z1$ python z1-bsl-nopic --z1 -c /dev/tt
yUSB0 -r -e -I -p print.ihex
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 2.13 (Device ID: f26f)
Changing baudrate to 38400 ...
Program ...
4809 bytes programmed.
Reset device ...
```

Figura 27[11]. Ejemplo de programa subiéndose a la mota.

Existe la posibilidad de comunicarse con las motas también desde el terminal, con el comando “make z1-reset & make login”, pero se encuentra con un problema de comunicación también con el puerto serie, el cual envía no solo lo que se le indique sino un espacio vacío detrás.

Así, para la comunicación con el Gateway vía puerto serie se ha utilizado el programa PuTTY.

PuTTY es un cliente SSH (entre otros tipos de comunicación, pero esta es la que me interesa) actualmente disponible en Windows y Linux.

Mediante la configuración del puerto serie donde tenemos el Gateway, dándole permisos mediante el comando:

$ sudo chmod 777 /dev/ttyUSB0

y fijando una velocidad de transacción (baud rate) de 115200, específica para el Zolertia Z1, es posible comunicarse con el dispositivo enviando y recibiendo mensajes del mismo:

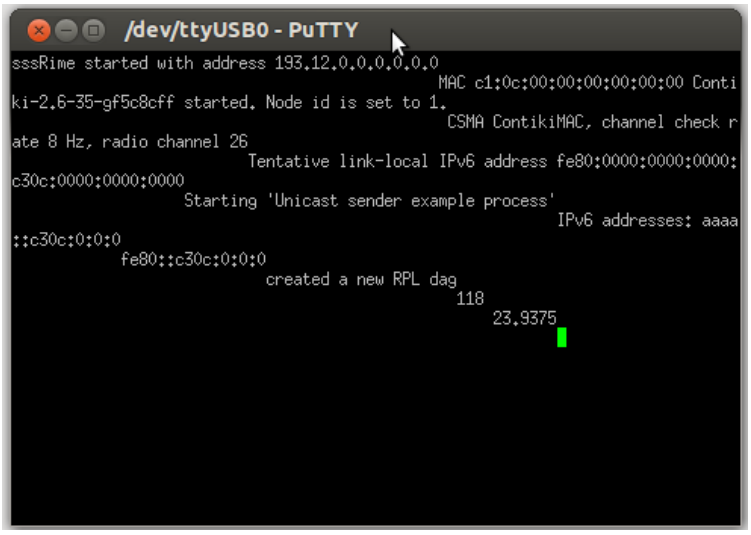


Figura 28. Inicio del Gateway visto desde el PuTTY, pidiendo función y temperatura.

El PuTTY ha sido muy útil en la realización de este proyecto para probar los comandos programados en las motas reales de una manera sencilla antes de incluirlos de forma definitiva en el interfaz de control.

CAPITULO 4

DISEÑO Y DESARROLLO DE LAS MOTAS

4.1 PUESTA A PUNTO (Contiki SO + Cooja) [11]

Aunque en apariencia puede parecer el paso más trivial de todos, la puesta a punto para utilizar Zolertia Z1 se encuentra diversos problemas.

Lo primero de todo es disponer de Ubuntu en un PC, o bien estar preparado para echar muchas horas delante de una máquina virtual del mismo. La elección para el proyecto fue probar ambos modos, para finalmente acabar quedándose con la primera solución.

A continuación se debe descargar el Contiki (nota: la versión de Contiki es diferente para la máquina virtual y para Ubuntu, y no funciona exactamente igual).

A continuación se conecta una mota y se prueba cualquiera de los muchos programas que Contiki nos ofrece como ejemplo. Y se encuentra lo siguiente:

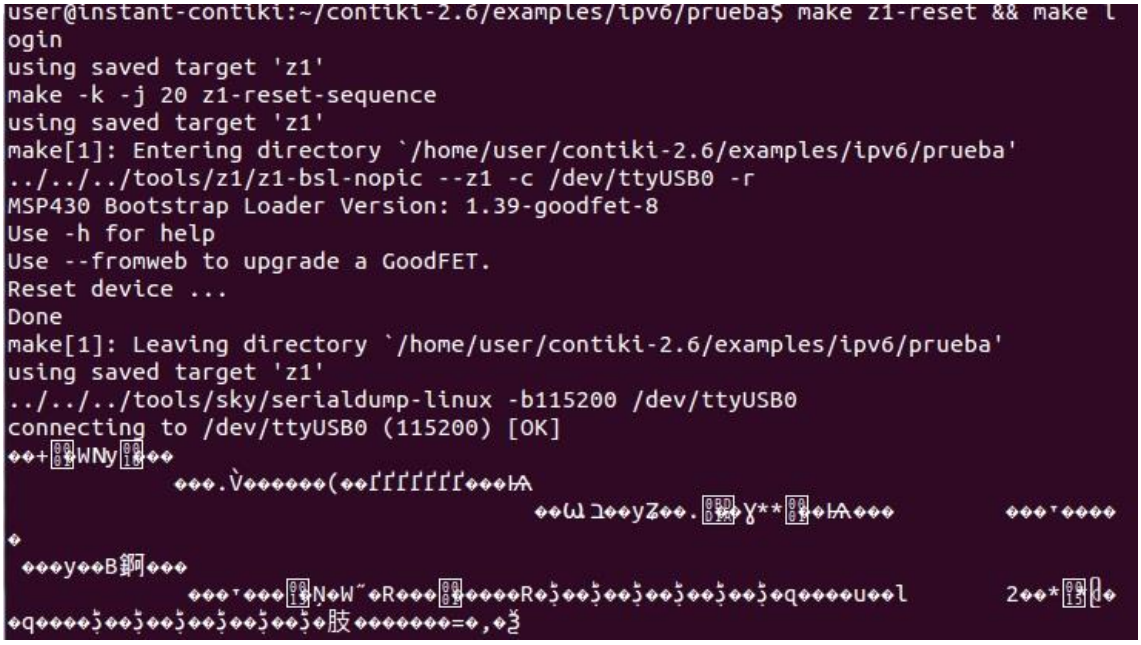


Figura 29. Este solo será el primero de los problemas con el que tenga que enfrentarse un desarrollador de Zolertia Z1.

Todo lo que el Zolertia Z1 devolvía era ruido.

Si vamos a la página de Zolertia para ver cuál es el problema, nos encontraremos con que dentro de varias páginas, dentro de una sola línea, nos dirá que “a veces” al utilizar una determinada versión de GCC (compilador), la 4.5.3, puede dar este error.

Que esto se soluciona con otra versión de GCC.

La verdad es que ese “a veces” resultó ser que de todas las personas que usan esta mota, a prácticamente todos nos sucedía, aunque a algunos, algunas veces (1 entre 4 o 5 reseteos) funcionaba bien.

Así pues, se puso en práctica la solución que recomienda Zolertia y descargamos el GCC 4.7.0, aun en beta. Después de varios pasos más (añadirlo al PATH, desinstalarlo varias veces, ya que se reinstalaba sola la versión anterior) se consigue tener el GCC 4.7.0 operativo, y tras recompilar los programas una vez más por fin se tiene la mota Z1 operativa.

```
user@instant-contiki:~/contiki-2.6/examples/z1$ make z1-reset && make login
make -k -j 20 z1-reset-sequence
make[1]: Entering directory `/home/user/contiki-2.6/examples/z1'
../../tools/z1/z1-bsl-nopic --z1 -c /dev/ttyUSB0 -r
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Use -h for help
Use --fromweb to upgrade a GoodFET.
Reset device ...
Done
make[1]: Leaving directory `/home/user/contiki-2.6/examples/z1'
../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Print num: 0
Print num: 1
Print num: 2
Print num: 3
Print num: 4
Print num: 5
Print num: 6
Print num: 7
Print num: 8
Print num: 9
^Cmake: *** [login] Interrupt
```

Figura 30. ¡Comunicación con el Zolertia Z1 conseguida!

El último problema que se encuentra a la hora de poner en funcionamiento la red de sensores, ya más avanzado el proyecto, reside en la asociación de IPv6 en función de la MAC (NODE-ID, número de identificación de la mota) que hace el dispositivo[12].

Debido a que de fábrica todos los dispositivos vienen con la misma configuración IP (terminada en 0:0) se necesita burnear los dos últimos octetos de la flash con un programa que viene incluido en Contiki. Excepto esos dos últimos octetos, todos lo demás de la MAC esta Hardcored.

Lo primero que hay que hacer para burnear la flash es ir a la localización del programa proporcionado con Contiki:

$ *Cd Contiki/examples/z1*

Con el siguiente comando:

```
$ make clean && make burn-nodeid.upload nodeid=X nodemac=X && make z1-reset && make login
```

y sustituyendo la X por el número que se le quiera dar al último octeto de la MAC, se puede diferenciar las motas.

En este caso se ha decidido dejar el Gateway terminado en 0, es decir, como venía de fábrica, y los dispositivos inalámbricos acabados en 11 (b) y 12 (c).

El output que debe dar el programa es:

```
Starting 'Burn node id'
Burning node id 158
Restored node id 158
```

Una vez realizado este cambio, al cargar cualquier programa que utilice funciones de red en la mota inalámbrica, en el inicio se debería ver lo siguiente:

```
Rime started with address 193.12.0.0.0.0.0.X
MAC c1:0c:00:00:00:00:00:X Contiki-2.5-1362-gd2aece8 started. Node id is set to X.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:X
```

Si bien todo esto se ha resumido en apenas dos hojas, representan muchas horas y tutorías de problemas y quebraderos de cabeza, limitándo la programación tan solo con el simulador ante la imposibilidad de hacerlo con la mota en sí.

4.2 CONFIGURACIÓN DEL GATEWAY

Para mostrar la configuración de las motas, tanto el Gateway como las que funcionan de forma inalámbrica, se ha realizado un resumen breve del programa, luego se irá comentando el código con el que han sido programadas y finalmente se presentará una tabla del resumen de los comandos.

Un resumen muy rápido del programa del Gateway sería el siguiente:

- El Gateway espera una petición de admisión a la red de parte de otra mota (inalámbrica), que tiene una IP y una función asociada.
- Una vez llega la petición, esta queda pendiente hasta que el Gateway decide si aceptarla o no. Solo puede haber una petición por vez, la última permanece.
- Si la petición es aceptada se integra la mota en la red y pueden empezar a realizarse comunicaciones con ella, asociar su botón a acciones en otras motas, pedir valores de sus sensores, etc...
- El Gateway también puede realizar todas estas funciones anteriores.

Se tratara de omitir en la medida de lo posible todos los conceptos y comandos que se vayan repitiendo de los diferentes programas excepto lo que sean necesario para la comprensión del conjunto del sistema.

En primer lugar se incluyen todas las librerías habituales, las necesarias para la comunicación inalámbrica vía IPv6-UDP, de comunicación vía puerto serie, de timers y de diversos sensores.

```
#include "contiki.h"
#include "sys/ctimer.h"
#include "sys/etimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "net/uip-debug.h"
#include "node-id.h"
#include "simple-udp.h"
#include "servreg-hack.h"
#include "dev/watchdog.h"
#include "net/rpl/rpl.h"
#include <stdio.h>
#include <string.h>
#include "dev/i2cmaster.h"
#include "dev/tmp102.h"
#include "dev/serial-line.h"
#include "dev/uart0.h"
#include "dev/leds.h"
#include "dev/button-sensor.h"
#include "dev/battery-sensor.h"
```

A continuación se definen algunas constantes útiles para el programa:

```
#define UDP_PORT 1234    //Puerto donde está el Gateway.
#define SERVICE_ID 190     // Service ID donde están las motas inalámbricas
#define SERVER_ID 189      // Service ID donde está el gateway.
#define SENSOR_READ_INTERVAL (3*CLOCK_SECOND)     //Tiempo entre lecturas del
//sensor de Luz.
#define NUMERO_MOTAS 5  // Número máximo de motas del sistema (modificable).
```

Para su uso tanto en el proceso principal del programa como en varios de las funciones necesarias (comunicaciones UDP o puerto serie), varias de las variables se han definido como globales a fin de facilitar su uso.

```
// Marca si hay coincidencia entre una IP que hace una petición y una ya existente en la
red.
int coincidencia=0;
// Sirve como contador para el FOR que realiza la función anterior.
int comprobador=0;
// Número actual de motas en la red.
 int contador=0;
//Sirve como contador para el FOR que realiza la función de carga desde fichero.
int contadorcarga=0;
// Marca si existe una petición de IP sin aprobar/rechazar.
int pcontador=0;
//  Sirve como contador para el FOR que ayuda a la eliminación de una mota de la red.
int bcontador=0;
// Indica la función de la mota que ha realizado la última petición de admisión a la red.
 int ptipo;
//Indica la función de las motas ya existentes en la red.
int tipo[NUMERO_MOTAS];
//Valor de la luminosidad de una mota.
static int pvalor;
//Valores de luminosidad de las motas ya existentes en la red.
static int valorlum[NUMERO_MOTAS];
//Direcciones IP de las motas ya existentes en la red.
static  uip_ipaddr_t addr[NUMERO_MOTAS];
//Dirección IP de la mota que ha realizado la última petición de admisión a la red.
 static uip_ipaddr_t paddr;
// Dirección IP a la que está asociado el botón del Gateway.
static uip_ipaddr_t baddr;
//Direcciones IP a las que están asociados los botones de las motas existentes en la red.
static uip_ipaddr_t biaddr[NUMERO_MOTAS];
//Acción a la que está asociado el botón del Gateway.
 char bact;
//Acciones a las que están asociados los botones de las motas existentes en la red.
```

```
 char biact[NUMERO_MOTAS];

//Comunicación proveniente del UART0, es decir, de la interfaz de usuario.
char var;
// Modelo de conexiones unicast.
 static struct simple_udp_connection unicast_connection;
```

A continuación se inicia el programa, que sigue el siguiente esquema:

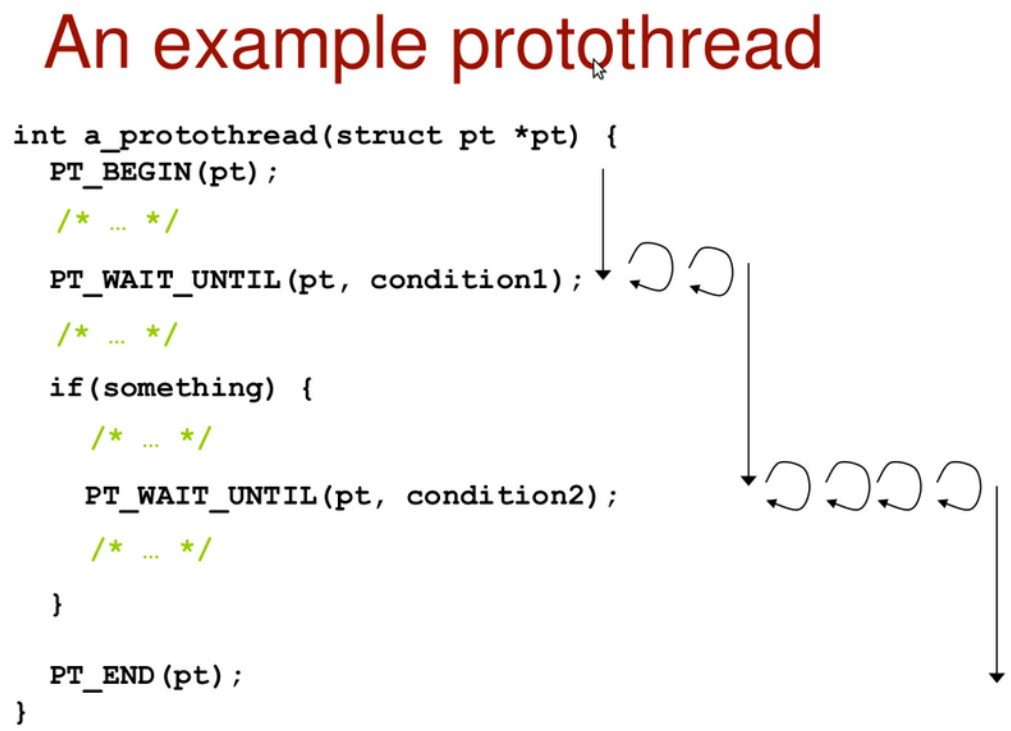


Figura 31[]. Esquema de un protothread.

Los protothreads son un mecanismo útil para la programación concurrente que funciona a base de procesos y eventos que despiertan dichos procesos.

Este tipo de esquema para la programación exige el uso de variables globales para que no se pierdan los valores en los cambios entre procesos.

-[] Figura 31.Iniciación a los protothreads. Sourceforge Zolertia(2010). Mainpage: Contiki Basics. Consultada el 15 de febrero de 2014 en: http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_Lesson_0

```
/*---------------------------------------------------------------------*/
PROCESS(unicast_sender_process, "Unicast sender example process");
AUTOSTART_PROCESSES(&unicast_sender_process);
/*---------------------------------------------------------------------*/
```

A continuación se programan algunas de las funciones externas al proceso principal.

La primera es static void receiver, que es la función por donde entran las comunicaciones inalámbricas.

Toda comunicación entrante recibirá una estructura de su composición, la dirección del envio, el puerto del envio, la dirección receptora, el puerto receptor, los datos propiamente y la longitud de dichos datos.

```
static void receiver(struct simple_udp_connection *c,
const uip_ipaddr_t *sender_addr,
uint16_t sender_port,
const uip_ipaddr_t *receiver_addr,
uint16_t receiver_port,
const uint8_t *data,
uint16_t datalen)
        {
        printf("%s",data);
```

Se muestran por pantalla los datos entrantes y a continuación, en función del dato, se trabaja con ellos. Los datos serán los provenientes de las respuestas que las motas inalámbricas hagan a determinadas peticiones del Gateway, como saber si la mota está activa, si se ha asociad correctamente la función, comprobación de la luminosidad de una mota, inversión de leds, etc..

Con esos nuevos datos se despierta el proceso principal con la función process_port para que realice las funciones que se le indiquen en la comunicación.

```
process_post(&unicast_sender_process, var, NULL);
```

Una función especialmente importante es la que recibe la petición de entrada a la red, dadas una IP y una función de la mota que realiza la petición.

La función comprueba si esta IP está ya en la red y si no está, confirma (NO acepta) la petición. Si existe una coincidencia, se acepta a la mota en la red ya que sería una mota que hubiese perdido la comunicación o se hubiese reseteado.

Según el dato que llegue con la petición se sabra la función de la mota que la está realizando.

```
if (*data == 'x' || *data == 'y' ||  *data == 'z' || *data == 'w' || *data == 'u')
```

Los sensores de luz enviarán una ‘z’, correspondiente a 122.

Los actuadores luminosos (LEDS) una ‘y’, correspondiente a 121.

Las motas inalámbricas de desarrollo con todas las funciones enviarán una ‘x’, correspondiente a 120.

La mota con la función interruptor enviará la función ‘w’, correspondiente a 119.

El Gateway se reserva para sí mismo la función ‘v’, correspondiente a 118.

El sensor de temperatura enviará una ‘u’, correspondiente a 117.

Para la comprobación de la IP se utiliza una función exclusiva de comparativa de IPs.

```
if(uip_ipaddr_cmp(&addr[comprobador], &*sender_addr))
```

Para realizar cualquier envió de vuelta a la mota, se usa la función simple_udp_sendto:

```
simple_udp_sendto(&unicast_connection, buf, strlen(buf) + 1, &addr[comprobador]);
```

A continuación se describen dos funciones básicas del SO Contiki usadas en el programa.

La primera función sirve para registrar la IPv6 en la red local que se está creando. Esta elección se hace en función de la MAC de cada mota, por eso es necesario “burnear” la memoria previamente con un número específico ya que de fábrica todas tienen la misma IPv6 asociada, terminada en 0.

La función devuelve la IP que se le ha asignado al dispositivo.

```
/*---------------------------------------------------------------------------*/
static uip_ipaddr_t *
set_global_address(void)
        {
        static uip_ipaddr_t ipaddr;
        int i;
        uint8_t state;
        uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
        uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
      uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
```

```
        printf("IPv6 addresses: ");
        for(i = 0; i < UIP_DS6_ADDR_NB; i++)
                {
                state = uip_ds6_if.addr_list[i].state;
                if(uip_ds6_if.addr_list[i].isused &&
                (state == ADDR_TENTATIVE || state == ADDR_PREFERRED))
                        {
                        uip_debug_ipaddr_print(&uip_ds6_if.addr_list[i].ipaddr);
                        printf("\n");
                        }
                }
return &ipaddr;
}
/*---------------------------------------------------------------------------*/
```

La última función sirve para registrar como RPL (protocolo de comunicación en función de cabeceras y demás) la IP que acabamos de crear. Esto permitirá la comunicación entre las motas que registremos como RPL.

RPL implementa una serie de prefijos a las comunicaciones que se realizan para que puedan ser entendidas entre las diferentes motas.

```
/*---------------------------------------------------------------------------*/
static void
create_rpl_dag(uip_ipaddr_t *ipaddr)
        {
        struct uip_ds6_addr *root_if;
        root_if = uip_ds6_addr_lookup(ipaddr);
        if(root_if != NULL) {
        rpl_dag_t *dag;
        uip_ipaddr_t prefix;
        rpl_set_root(RPL_DEFAULT_INSTANCE, ipaddr);
        dag = rpl_get_any_dag();
        uip_ip6addr(&prefix, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
        rpl_set_prefix(dag, &prefix, 64);
        PRINTF("created a new RPL dag\n");
        }
else
        {
        PRINTF("failed to create a new RPL DAG\n");
        }
}
/*---------------------------------------------------------------------------*/
```

Seguidamente se ha programado la función ultimooctet o.

Las IPv6 están formadas por 128 bits, repartidos hexadecimalmente, así pues tomando el octeto número 15 de la dirección se conocerá el último dígito de la misma

La función devuelve eso mismo, el último octeto de la IP que se le pase.

```
/*---------------------------------------------------------------------------*/
uint16_t ultimoocteto(const uip_ipaddr_t *addr)
        {
        uint16_t a;
        a = addr->u8[15];
        return ( a);
         }
/*---------------------------------------------------------------------------*/
```

La última función previa al proceso principal sirve para ver el comando que entra por el UART0, es decir, la comunicación con el puerto serie. Uart_rx_callback representa un evento de entrada de datos por ese mismo puerto. Luego con Process_post se vuelve de nuevo al proceso principal, ya que interesa que tras cada entrada de datos por el UART0 proceda en función de lo recibido.

```
static int uart_rx_callback(unsigned char c)
        {
        var = c;
        process_post(&unicast_sender_process, var, NULL);
        }
```

A continuación se entra en el proceso principal del programa, durante el cual se usan todas las comunicaciones que he indicado en las funciones anteriores, tanto las provenientes del puerto serie como las inalámbricas, para realizar diferentes funciones.

El esquemático general del programa es el indicado en la figura 31, y consiste en un bucle indefinido que funciona con eventos, ya sea pulsar el botón o la entrada de datos por puerto serie o vía inalámbrica.

Declaración de algunas variables e inicio del proceso.

```
uip_ipaddr_t *ipaddr; // IP del Gateway.
static struct etimer et; // Timer para algunos procesos.
PROCESS_BEGIN();
int accion=0; //Comprobante para usar la función asociada al botón del Gateway.

//Variables para la medición de la temperatura:
int16_t tempint;  uint16_t tempfrac; int16_t raw;  uint16_t absraw; int16_t sign; char minus = ' ';
```

```
static int d=0; // Variable que marca la mota seleccionada actualmente.
unsigned char ultimooct = ' '; // Último octeto de una IP.
```

A continuación se inicializan algunas funciones importantes de la mota:

```
servreg_hack_init(); //  Se inicia el servicio de registro de IPs del área local.
ipaddr = set_global_address(); // Se fija la variable ipaddr como la dirección IP del //Gateway.
create_rpl_dag(ipaddr); //Se fija nuestra IP para protocolo RPL.
servreg_hack_register(SERVER_ID, ipaddr); // Se registra nuestra IP en el área local, //en el puerto SERVER_ID, donde solo estará el Gateway.
simple_udp_register(&unicast_connection, UDP_PORT,NULL, UDP_PORT, receiver); // Se registra nuestra mota para la comunicación UDP.
tmp102_init(); // Se inicia el sensor de temperatura.
uart0_init(BAUD2UBR(115200)); //Se inicia el uart0 para las comunicaciones del //puerto serie.
uart0_set_input(uart_rx_callback); //Cuando llegue algo, se llama a la función //UART_RX_CALLBACK definida anteriormente
SENSORS_ACTIVATE(battery_sensor); // Se activa el sensor de bacteria.
SENSORS_ACTIVATE(button_sensor); //Se activa el botón.
```

Y por último se entra en el bucle indefinido donde estará el programa. El bucle hará una vuelta a cada variación de la variable **var,** que nos indica que se ha recibido algún tipo de comunicación. Se invierten los LEDS en cada vuelta para visualizar el proceso.

```
leds_invert(LEDS_GREEN);
```

El proceso espera un evento, ya sea de botón o de entrada de una orden por el uart0.

```
PROCESS_WAIT_EVENT();
```

Dentro de los varios procesos que se pueden pedir al mismo Gateway, está el de medir la temperatura, que se realiza con la siguiente función:

```
raw = tmp102_read_temp_raw();
```

Otros procesos serían similares a los que se les pide a las motas inalámbricas, como pedir el estado, la función que desempeña, etc..

Algunas funciones están reservadas para pruebas, por funciones que no han resultado correctas o para el desarrollo. Una de ellas es un ejemplo de comunicación unicast básico. Se crea un buffer, se llena con lo que se quiere enviar con la función sprintf, se

visualiza la IP de destino con uip_debug_ipaddr_print y finalmente lo se envía con el ya explicado simple_udp_sendto:

```
if(var == 101)
        {
        char buf[20];
        printf("Sending unicast to ");
        uip_debug_ipaddr_print(&addr[contador-1]);
        printf("\n");
        sprintf(buf, "b");
        simple_udp_sendto(&unicast_connection, buf, strlen(buf) + 1, &addr[contador-
1]);
        uip_debug_ipaddr_print(&addr[contador-1]);
        }
```

Las mismas funciones y alguna más si se dispone de sensores externos) que se le piden al Gateway, se le pueden pedir también a las motas inalámbricas que han sido aceptadas en la red.

Con el siguiente comando se puede ver cuando batería les queda disponibles a las motas, algo que también puede hacerse con el Gateway pero que no tiene la misma importancia, ya que el Gateway tiene alimentación vía USB por estar conectado al PC y las motas inalámbricas no.

```
uint16_t bateria = battery_sensor.value(0);
```

Para el proyecto se intentó diseñar una función que cambiase la IP dentro del mismo programa del Gateway para así ir concediendo un número de IP según la entrada a la red y no según la MAC de cada dispositivo. Esta función NO funciona.

```
watchdog_stop();
leds_on(LEDS_RED);
int NODEID=1;
//#if NODEID
// #warning "***** BURNING NODE ID"
printf("Burning node id %d\n", NODEID);
node_id_burn(NODEID);
leds_on(LEDS_BLUE);
node_id_restore();
printf("Restored node id %d\n", node_id);
//#else
//#error "burn-nodeid must be compiled with nodeid=<the ID of the node>"
// node_id_restore();
```

```
// printf("Restored node id %d\n", node_id);
//#endif
leds_off(LEDS_RED + LEDS_BLUE);
watchdog_start();
```

La siguiente función fue diseñada con el fin de poner conocer el estado de los LEDS y así poder no solo invertirlos sino trabajar con ellos en ON y OFF. Esta función no está habilitada para el Contiki de la versión Linux, solo para el de la versión de la máquina virtual. Solo leds_invert funciona.

```
//Funcion de prueba de inversión controlada de LEDS. No funciona.
if(var == 117)
        {
        leds_invert(LEDS_ALL);
        //leds_on(LEDS_RED);
        // leds_off(LEDS_RED + LEDS_BLUE + LEDS_GREEN);
        //PRINTF("s\n");
        }
```

Para la función de carga desde fichero, se puede construir una IP a partir de la función siguiente, y modificando el último octeto de la misma (que será el dato guardado en el fichero), se podrá reponer toda la red aun después de un reinicio.

Ejemplo para IP terminada en b (u 11):

```
ultimooct = 0xb;
uip_ip6addr(&baddr,  0xaaaa,0,0,0,0xc30c,0,0,ultimooct);
printf ("Dirección definida: ");
uip_debug_ipaddr_print(&baddr);
```

Para la asociación de funciones o de botones con otros dispositivos, se espera un serie de comandos que provienen de la interfaz de control y se fijan una dirección IP y una acción que realiza el botón.

A partir de ahí, cualquier pulsación hará que se entre en el bucle indefinido y salte directamente a la parte de la acción del botón.

```
//p. Asociación del botón del Gateway con un led de otra mota(o de si //mismo).
if(var == 112)
{
var=NULL;
//printf("Esperando 1");
```

```
PROCESS_WAIT_UNTIL(var != NULL);
if(var == 97 || var == 98 || var ==99 )
        {
        if ( var == 97)
                {
                baddr = *ipaddr;
                }
        if ( var == 98)
                {
                baddr = addr[0];
                }
        if ( var == 99)
                {
                baddr = addr[1];
                }
        var=NULL;
        //printf("Asociado a:");
        //uip_debug_ipaddr_print(&baddr);
        //printf("\n");
        //printf("Esperando 2");
        PROCESS_WAIT_UNTIL(var != NULL);

        if(var == 114 || var == 115 || var ==116 )
        {
        if(var == 114)
        {
        bact='r';
        }
        if(var == 115)
        {
        bact='s';
        }
        if(var == 116)
        {
        bact='t';
        }
        //printf("Asociado a comando: %c \n", bact);
        }
printf("o");
}
```

Acción que se realiza si en vez de llegar una orden por el UART0, se pulsa el botón, es decir, se realiza la acción que antes se haya asociado al botón del Gateway:

```
Else
        {
        if (accion==0)
                {
                char buf[20];
                printf ("Enviado a la dirección predefinida: ");
                uip_debug_ipaddr_print(&baddr);
                printf("Comando: %c \n", bact);
                sprintf(buf,"%c", bact);
                simple_udp_sendto(&unicast_connection, buf, strlen(buf) + 1,
&baddr);
                accion=1;
                }
        }
```

Y se finaliza el proceso y con ello el programa del Gateway.

```
PROCESS_END();
/*---------------------------------------------------------------------------*/
```

TABLA RESUMEN DE COMANDOS DEL GATEWAY.

Comando (ASCII)	Gateway	Mota Inalámbrica
a (97)	Pide estado.	Pide estado.
b (98)	Pide temperatura.	Pide temperatura.
c (99)	Pide función.	Pide función (*).
d (100)	Función de prueba.	
e (101)	Función de prueba.	
f (102)	Pide IP dispositivo.	Pide IP dispositivo (*).
g (103)	Indica mota 1.	
h (104)	Indica mota 2.	
i (105)	Indica que vuelve a dirigirse al Gateway.	
j (106)	Pide IPs de todos los dispositivos asociados.	
k (107)	Pide número de dispositivos asociados.	
l (108)	Abre los puertos del Gateway.	Pide luminosidad.
m (109)	Pide batería.	Pide batería.
n (110)	Función de prueba de burneo. No funciona.	
o (111)	Borra la mota actual de la red.	
p (112)	Asocia botón.	Asocia botón.
q (113)	Función de prueba.	
r (114)	Invierte Led verde.	Invierte Led verde.
s (115)	Invierte Led rojo.	Invierte Led rojo.
t (116)	Invierte Led azul.	Invierte Led azul.
u (117)	Función de prueba de LEDS. No funciona.	
v (118)	Pide detalles de la última petición de entrada a la red.	
w (119)	Acepta la última petición de entrada a la red.	
x (120)	Rechaza la última petición de entrada a la red.	
y (121)	Realiza la carga desde fichero.	

Tabla 3. Todos los comandos de entrada por el UART0. (*) Aunque devuelva información de una mota inalámbrica, la proporciona el Gateway.

FLUJOGRAMA REDUCIDO DEL GATEWAY.

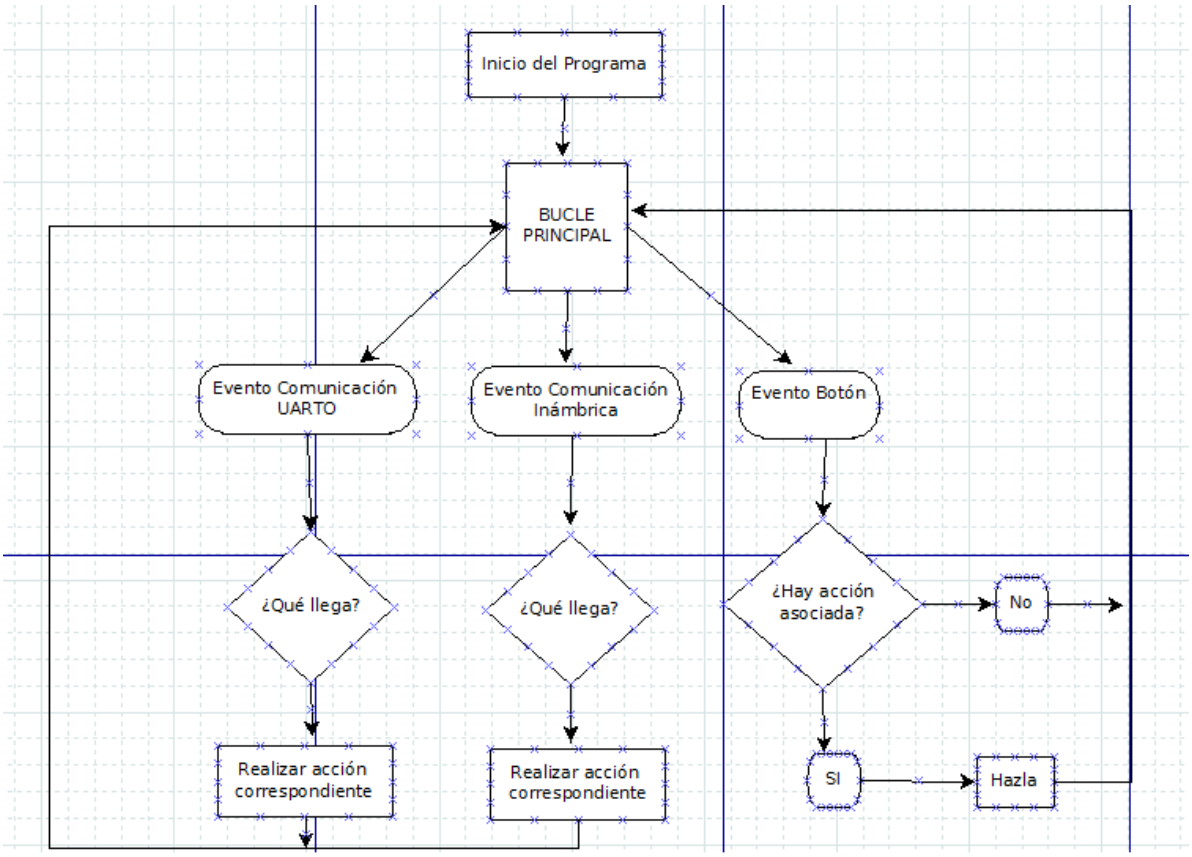


Figura 32. Flujograma del Gateway.

Como resumen del modo de funcionamiento del Gateway, en este flujograma se puede ver, sin entrar en detalles sobre que peticiones concretas se hace en el programa, el modo de funcionamiento del mismo. Destacan los tres tipos de eventos , Comunicación UART, Comunicación Inalámbrica y Botón que hacen que el programa principal realice alguna acción.

4.2 CONFIGURACIÓN DE LOS DISPOSITIVOS PERIFÉRICOS

Los dispositivos inalámbricos, también Zolertia Z1, han sido programados de una forma diferente a la del Gateway.

Durante la realización del proyecto se ha trabajado con una mota (previa programación) que disponía de todo tipo de funciones. Para la finalización y simulación "realista del proyecto" se han programado varias motas mono función que serían más acordes al objetivo del proyecto de llevarse a cabo el mismo.

Los sensores y actuadores inalámbricos tienen variaciones respecto al Gateway en su programación.

Mientras que el grueso de tipos de comunicaciones del Gateway se encuentra en la entrada por el puerto serie y recibe un número pequeño de comunicaciones de forma inalámbrica, los dispositivos periféricos trabajan al contrario. Al no tener conexión vía puerto serie, su grueso de comunicaciones se traslada al aspecto inalámbrico.

Para la puesta en marcha de las funciones inalámbricas de estas motas se emplean las mismas funciones que en el Gateway, así se ve repetido:

```
static void receiver
static uip_ipaddr_t *set_global_address(void)
static void create_rpl_dag(uip_ipaddr_t *ipaddr)
```

La parte donde más se diferencia la programación del Gateway de la del resto de motas inalámbricas es en la inicialización del mismo. Se registra la mota en SERVICE_ID, definido como 190:

```
#define SERVICE_ID 190
#define SERVER_ID 189
servreg_hack_register(SERVICE_ID, ipaddr);
```

Las motas tienen 2 bucles en vez de uno, ya que el primero requiere la espera desde que se realiza la petición de entrada a la red hasta que es aceptada. Con la siguiente función se localiza al Gateway, ya que será la única mota inscrita en SERVER_ID, mientras todas

las demás estarán en SERVICE_ID, y se le asocia a la dirección del envío de petición de entrada en la red.

El envío (en este caso una “x”, perteneciente a una mota inalámbrica de desarrollo con todas las funciones) puede variar en función del tipo de programa que contenga el dispositivo inalámbrico.

```
addr = servreg_hack_lookup(SERVER_ID);
if(addr != NULL)
        {
        char buf[20];
        printf("Estableciendo conexion con servidor: \n");
        uip_debug_ipaddr_print(addr);
        printf("\n");
        sprintf(buf,"x");
        simple_udp_sendto(&unicast_connection, buf, strlen(buf) + 1, addr);
        printf ( "Esperando respuesta\n");
        etimer_set(&rt, SEND_INTERVAL);
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        }
```

En caso de recibir respuesta del Gateway, la mota entrará en estado activo.

Aquí hay que hacer una nueva diferenciación entre los diferentes tipos de motas inalámbricas:

-Para los dispositivos que funcionen como actuadores, entrar en el estado activo será su estado final, quedando a disposición de ser contactados por el Gateway para cualquier requerimiento y también en disposición de ser asociados a una mota funcionando como sensor.

-Por otro lado los dispositivos que funcionen como sensores, una vez entrado en estado activo aún no estarán plenamente operativos hasta que no se les asocie con un dispositivo con función de sensor.

FLUJOGRAMA REDUCIDO DE UNA MOTA INALÁMBRICA EN FUNCIÓN DE ACTUADOR.

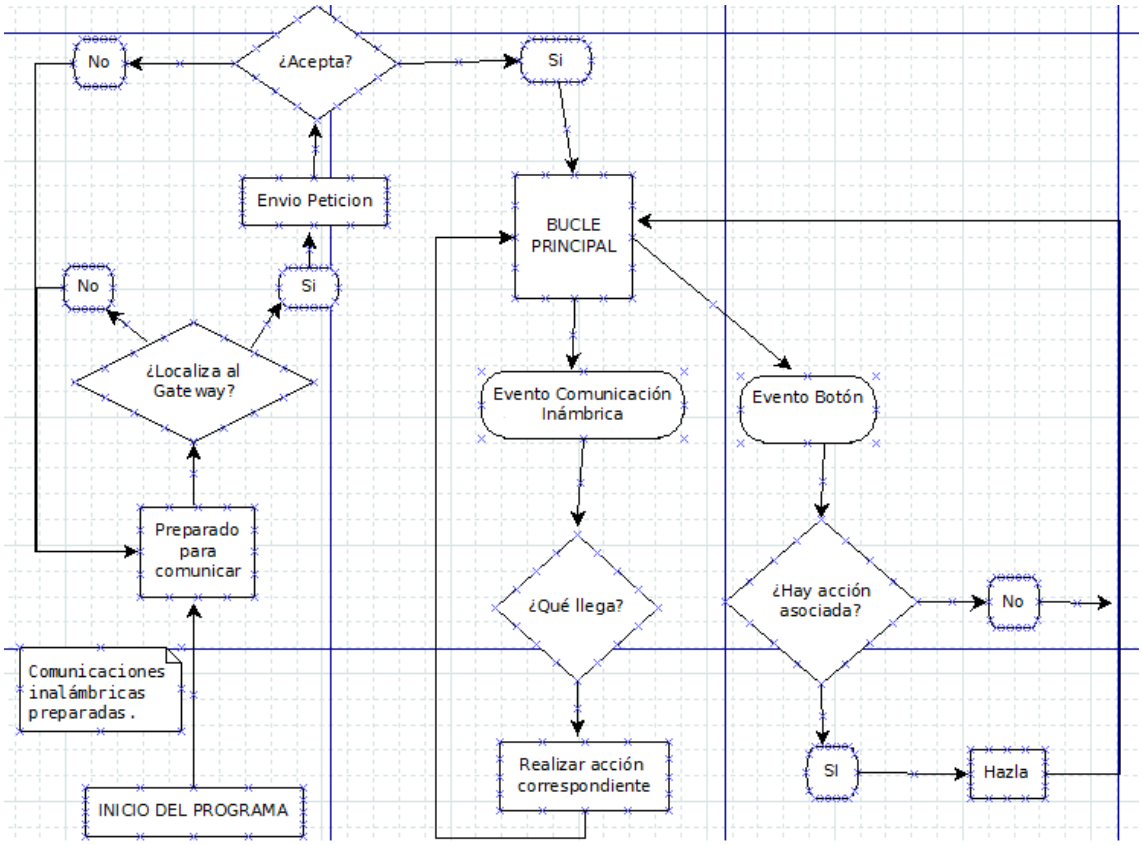


Figura 33. Flujograma de un actuador.

El flujograma de una mota inalámbrica en función de actuador es similar en el aspecto del bucle principal a la del Gateway pero tiene es muy diferente en la parte inicial del programa. Hasta el momento en el que no se ha realizado la comunicación satisfactoria con el Gateway para realizar una petición de ingreso a la red, y ser aceptada, no entrará en el funcionamiento normal.

FLUJOGRAMA REDUCIDO DE UNA MOTA INALÁMBRICA EN FUNCIÓN DE SENSOR.

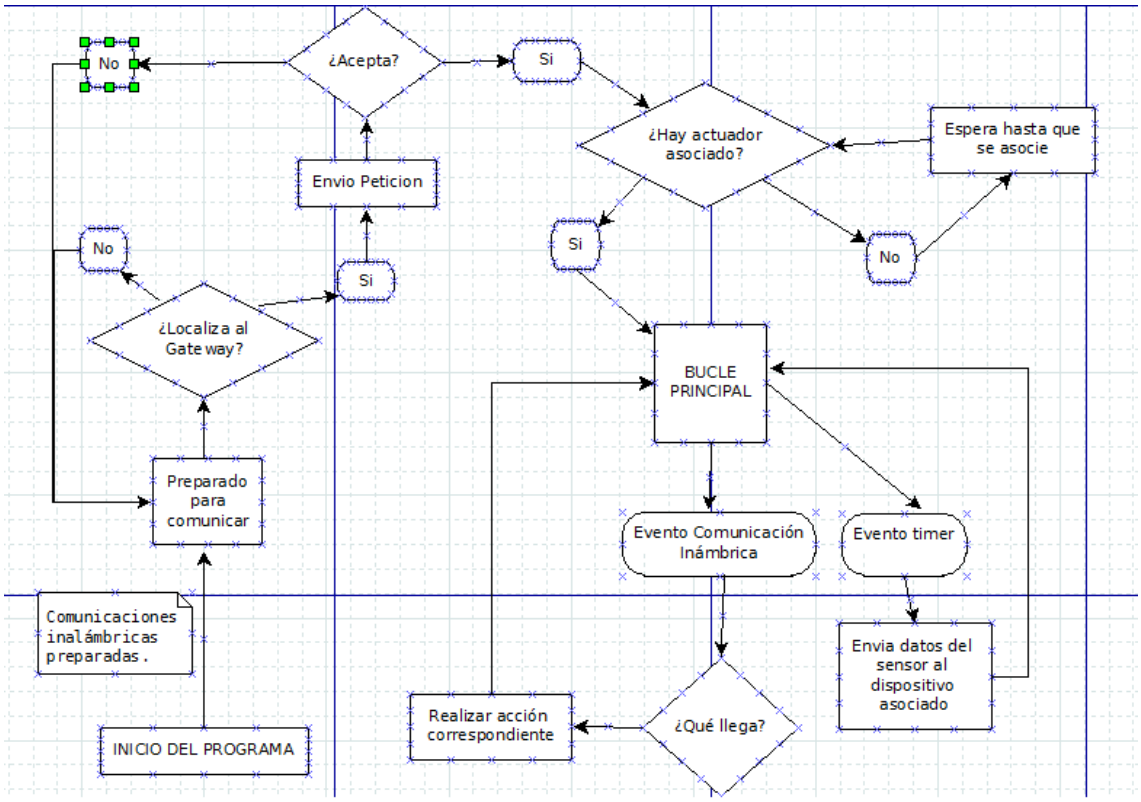


Figura 34. Flujograma de un sensor.

El flujograma de una mota inalámbrica en función de sensor guarda pocas semejanzas con el Gateway.

Su proceso principal es el envío cada cierto tiempo (evento del timer) de la medición que realice su sensor (ya sea temperatura, luminosidad,etc…).

A su vez puede seguir respondiendo a peticiones del Gateway dentro de su bucle principal.

Respecto al inicio del programa, se diferencia del flujograma de los actuadores en que previamente a iniciar el bucle principal se pide estar asociado a una mota, para poder enviar la información del sensor.

Uno de los puntos más complicados de las comunicaciones entre dispositivos es el trabajo con los tipos de datos. Simple_udp_sendto (función de envío de información) es

una función que admite un tipo de dato y receiver (función de recepción de información) recibe otro. Como en estos casos es importante ser cuidadosos con la memoria disponible a veces se trata con tipos de datos de difícil manejo.

Un ejemplo es el paso de temperatura, donde partiendo de uint16_t se tiene que convertir primero a uint8_t y luego "deshacer" el ASCII para tener los valores desde los que se ha partido trabajando con la memoria:

```
const uint16_t *data pasamos a uint8_t:
```

se pasa a uint8_t:

```
partB[1] = (uint8_t)((data[1] & 0xFF00) >> 8);
partA[1] = (uint8_t)(data[1] & 0x00FF);
```

y finalmente, restandole 48 (ya que llega como char, y se necesita el int que fue previamente):

```
intdatoentrada[1] = partA[1]-48;
```

Como este ejemplo son numerosas las transformaciones de datos que tienen que darse ya que todo es enviado de una forma por un dispositivo pero llega de otra al receptor, teniendo que volver a adecuarlo.

CAPITULO 5

INTERFAZ DE CONTROL (QT CREATOR)

5.1 CONFIGURACIÓN

El interfaz de control del usuario se ha realizado con la aplicación QtCreator.

La base principal del programa es la comunicación puerto serie (PC) – UART0 (Gateway) en ambos sentidos, ya que la idea no es solo transmitir ordenes al Gateway para configurar la red de sensores y actuadores sino recibir la información de este y poder mostrarla de una forma más agradable visualmente.

La interfaz de usuario cuenta con un pequeño tutorial, para usuarios novatos, o puede derivarse directamente al esquemático de las conexiones para acceder directamente a la configuración de los distintos dispositivos.

Además, se puede modificar la configuración para comunicarse con los diferentes puertos series del ordenador, ya que el Gateway no tiene por qué estar necesariamente (aunque suele ser así) conectado al ttyUSB0.

Figura 35. Inicio de la interfaz de usuario.

Para toda la comunicación vía Puerto Serie con el Gateway se ha usado una función fundamental en el programa, haciendo uso de la librería `<SerialPort.h>`, este sería un ejemplo de uso, de petición de la temperatura:

```
float comunicacion::pedirtemperatura()
{

SerialPort cpserie(cpuerto);


//-- Abriendo el puerto serie con la configuraciÃ³n: 115200 baud, 8N1

cout << "Puerto Serie: " << cpuerto << endl;
cout << "Abriendo..." << endl;

try
        {
        cpserie.Open(SerialPort::BAUD_115200,
        SerialPort::CHAR_SIZE_8,
        SerialPort::PARITY_NONE,
        SerialPort::STOP_BITS_1,
        SerialPort::FLOW_CONTROL_NONE);
        }

catch (SerialPort::OpenFailed E)
        {
        cout << "Error opening the serial port" << endl;
        }
// Se crea un buffer para ir leyendo lo que el Gateway escribe en el puerto serie.

string envio = "b";
SerialPort::DataBuffer buffer;
cpserie.Write(envio);
float temperatura[5];
cout << "Esperando valores..." <<endl;
try
        {
        cpserie.Read(buffer,9,10000);
        }
catch (SerialPort::ReadTimeout E)
        {
        cout << "TIMEOUT!, Por favor, comprueba la conexiÃ³n.";
        }
```

Los valores del Buffer son tomados en ASCII así que se transforman dichos valores a integer restándole 48 a los números.

```
cout << "Valores tomados." << endl;
cout << endl;
cout <<buffer[0]<<endl;
cout <<buffer[1]<<endl;
cout <<buffer[2]<<endl;
cout <<buffer[3]<<endl;
cout <<buffer[4]<<endl;
cout <<buffer[5]<<endl;
cout <<buffer[6]<<endl;
cout <<buffer[7]<<endl;
cout <<buffer[8]<<endl;
temperatura[0]=buffer[0]-48;
temperatura[1]=buffer[1]-48;
temperatura[2]=buffer[2]-48;
temperatura[3]=buffer[3]-48;
temperatura[4]=buffer[4]-48;
temperatura[5]=buffer[5]-48;
//Finalmente  se construye la temperatura final.
float total = temperatura[1]*10+temperatura[2]+temperatura[4]*0.1+
temperatura[5]*0.01;
cout << total << endl;
return(total);

}
```

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Tabla 4[]. Tabla de correspondencia decimal-hexadecimal-octal-HTML-char (ASCII).

-[]Tabla de correspondencia ASCII. Asciitable (2010). ASCII table and description. Consultada el 20 de febrero de 2014 en: http://www.asciitable.com/

En general, el procedimiento, aunque no los comandos que se le envían y los que se espera recibir, es siempre el mismo:

-Se envía un comando mediante el puerto serie que hará que el Gateway cumpla alguna función e imprima (o no) algo por pantalla.

-Se lee el puerto serie para ver la respuesta, de la cual se sabe el formato y el tamaño, y lo se asocia con los datos que interese mostrar al usuario.

Además de la comunicación con el Gateway, el interfaz de usuario también es el encargado de guardar determinados datos para hacer una carga en el caso de que se produzca un reinicio de Gateway. Los datos guardados son:

-Número de motas en la red.

-IP a la que está asociado el botón del Gateway.

-Acción a la que está asociado el botón del Gateway.

-IP de la mota inalámbrica 1.

-Función de la mota inalámbrica 1.

-IP de la mota inalámbrica 2.

-Función de la mota inalámbrica 2.

-Etc..

Con esto se asegura poder hacer una carga rápida de la red de sensores y actuadores si el Gateway es reiniciado o si existe algún problema.

El código de guardado y de carga es el siguiente:

```
//------------------------ GUARDANDO DATOS ------------------------//

if(pagina == "guardarcasa")
{
ofstream fichero("Casa guardada.txt"); // crear o rescribir archivo

cout << "NÃºmero Dispositivos: " << e.getdispositivo() << endl;
fichero << e.getdispositivo() << endl;
cout << "Ultimo Direccion IP Boton: " << cont.getultimoipboton() << endl;
fichero << cont.getultimoipboton() << endl;
cout <<"Ultimo Comando Boton: " << cont.getbact() << endl;
```

```
fichero << cont.getbact() << endl;
for (contador = 1; contador <=e.getdispositivo();contador++)
        {
        cout <<"IP de la mota numero " << contador << "guardada como : " << mota[contador].getultimoip() << " y funcion"<< mota[contador].getfunc() << endl;
        fichero << mota[contador].getultimoip() << endl;
        fichero << mota[contador].getfunc() << endl;
        }


// Verifica la creaciÃ³n del archivo
if ( fichero.bad() )
        {
        cout << "Error al tratar de abrir archivo";
        cin.get();
        }

fichero.close();
cout << "archivo creado exitosamente" << endl;
pagina="esquematico";
contador = 1;
}
```

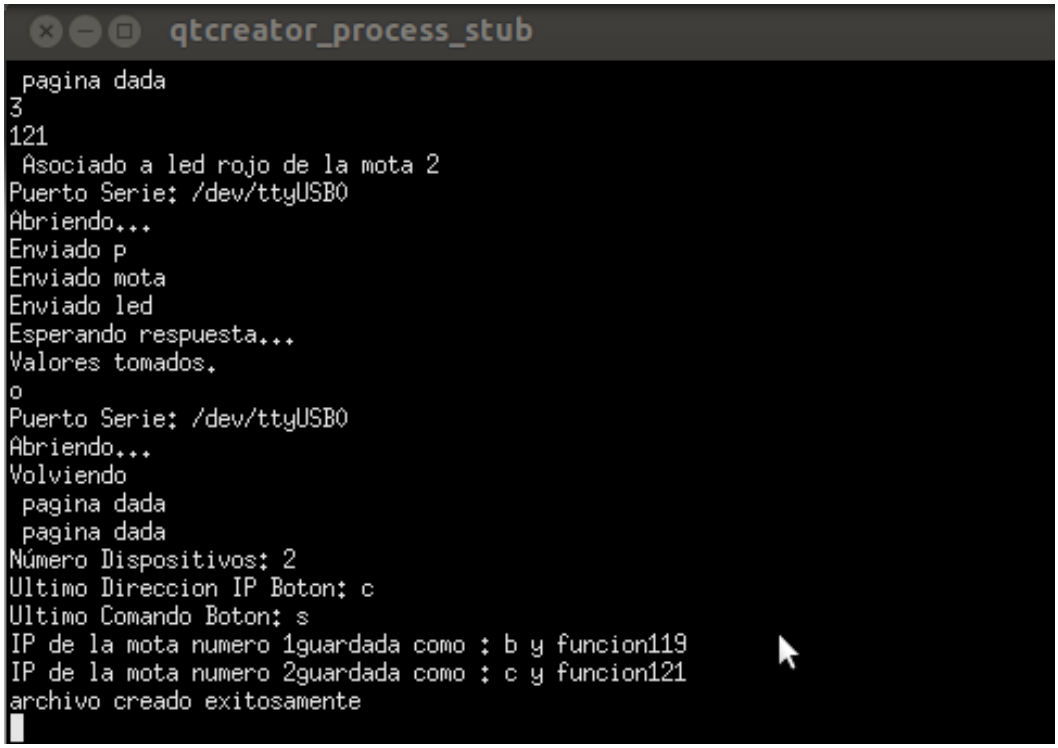


Figura 36. Guardado de datos visto desde el terminal.

```
//------------------------ CARGANDO DATOS ------------------------//
if(pagina == "cargarcasa")
        {
        cout << "probando" << endl;
        char cadena[2]="";
        ifstream fe("Casa guardada.txt", ios::in);
        if(fe.good())
                {
                fe >> cadena;
                int dispositivos;
                dispositivos= atoi(cadena);
                cout << dispositivos << endl;

                fe >> cadena;
                char diripbot[2] ;
                strcpy(diripbot, cadena);
                cout << diripbot << endl;

                fe >> cadena;
                char combot[2] ;
                strcpy(combot, cadena);
                cout << combot << endl;


                for (contador = 1; contador <=dispositivos; contador++)
                        {
                        fe >> cadena;
                        strcpy(dispo[contador], cadena);
                        cout << dispo[contador] << endl;

                        fe >> cadena;
                        funcion[contador]=atof(cadena);
                        cout << funcion[contador] << endl;
                        }


                c.cargarcasa(dispositivos, diripbot, combot , dispo[1],funcion[1],
dispo[2],funcion[2]);
                }

        else
                {
                cout<<"\nNo hay ninguna casa guardada\n"<<endl;
                if(fe.fail()) cout << "Bit fail activo" << endl;
                if(fe.eof()) cout << "Bit eof activo" << endl;
                if(fe.bad()) cout << "Bit bad activo" << endl;
```

```
                }

        pagina="esquematico";
        contador= 1;
        }
```

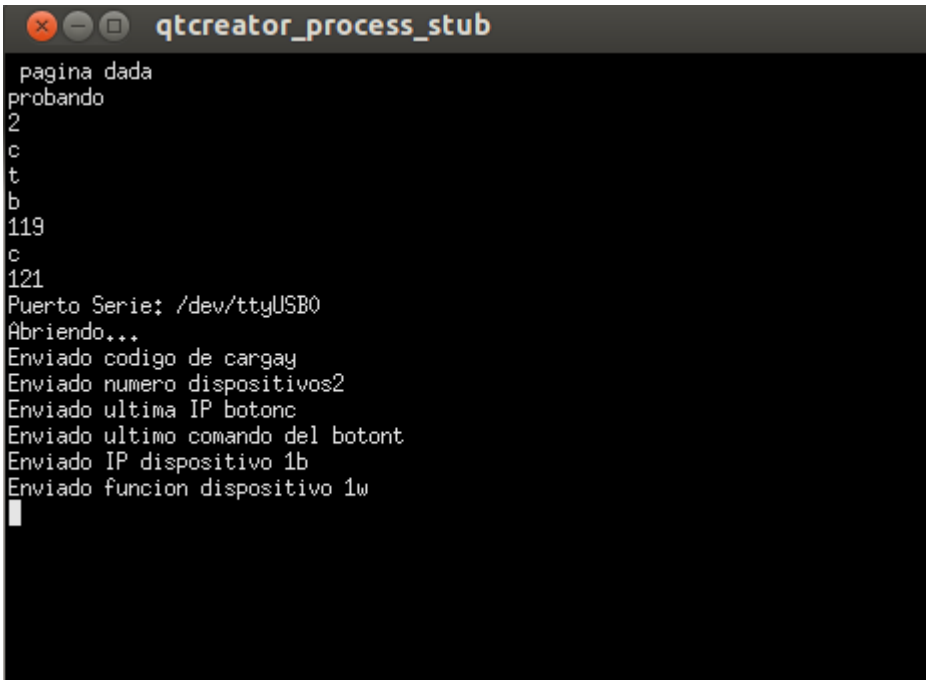


Figura 37. Carga de datos visto desde el terminal.

Toda esta carga y descarga se hace mediante ficheros externos al programa, escribiendo letras o números clave y luego interpretándolos.

Es decir, con el interfaz de usuario se tiene un acceso rápido y visualizado de todas las funciones que realiza el Gateway y algún extra para facilitar el guardado de información y la comunicación puerto serie.

Además de estos aspectos técnicos, se ha intentado realizar un diseño visual agradable y intuitivo para el manejo y configuración de las motas, además de paneles individuales de cada mota en los que se puede asociar y ver los datos, de forma similar al Gateway, también se cuenta con un Panel de Control para tener una visión más global del sistema.

Todo el diseño virtual se ha realizado mediante Widgets, que proporcionan una variedad de librerías y componentes muy vistosos para la programación de una aplicación para usuarios no exclusivamente técnicos.

La programación de los Widgets varía de dificultad, desde sencillos botones colocados de forma visual y programados en código, por ejemplo:

```
void ControlDispositivo::on_bateria_clicked()
```

Que a su vez pueden producir cambios en displays y labels, o llamar a funciones externas al Widget.

```
bateria = punt1->pedirbateria();
ui->numbat->display(bateria);
```

La comunicación entre las diferentes clases del programa , al ser Widgets, no queda más remedio que hacerla mediante punteros.

En este caso se ha definido una clase "Gestor Comunicación" que se encarga de acumular toda la información del programa para devolverla en el momento que haya necesidad de ella (GETs y SETs).

También pueden realizarse montajes algo menos sencillos con los Widgets, como en el caso de los QComboBox, que sirven para crear listados, tanto predefinidos como en función de eventos. Con la siguiente función enlazamos lo que aparece en un QComboBox en función de lo que se haya seleccionado en otro:

```
void ControlDispositivo::on_eleccionmota_activated(int index)
```

Finalmente, pueden realizarse cosas bastante complejas como la implementación de todo en Widget en función de eventos que vayan ocurriendo a su alrededor.
Esta opción ha sido la elegida para realizar el Panel de Control.
Usando QGroupBox se puede facilitar el manejo de un Widget con una cantidad de información importante.
Lo primero de todo, se incluye la librería

```
#include <QGroupBox>
```

A continuación se declara el QGroupBox y todos los elementos que vaya a llevar dentro. En este caso se usa un array de 6 QGroupBox y de todos esos elementos ya que es el máximo de motas para las que está preparada cada hoja del panel de control, y en este caso solo puede llenarse una (1 Gateway + 2 Motas Inalámbricas + 3 Motas Virtuales).

```
QGridLayout *gbox[6];
QVBoxLayout *vbox[6] ;
QLabel *Funcion[6];
QLabel *Imagen[6];
QLabel *Asociado[6];
QLabel *Asociadocon[6];
```

Además de la QGroupBox, definida como varible privada de la clase, y las labels que llevará dentro, se han definido 2 modelos de Layout de Widgets (posteriormente solo se ha usado hoy).

QVBoxLayout dispone todos los elementos que se le relaciones de manera vertical.

Exactamente igual que QVBoxLayout funciona QHBoxLayout, pero horizontalmente.

La solución elegida ha sido QGridLayout, que solicita no solo la lista de elementos sino una división de la QGroupBox en final y columnas, dividiendo su espacio entre los elementos proporcionados.

Es posible cambiar el título general de la QGroupBox (llamada grupo[i]), que se coloca en la parte superior izquierda:

```
grupo[i]->setTitle("Mota "+ QString::number(i));
```

Con la siguiente función se situa geometricamente el QGroupBox dentro del Widget y sus dimensiones, en este caso X e Y han sido previamente definidas para ir variando según se van añadiendo nuevas motas.

```
grupo[i]->setGeometry(x,y,261,171);
```

Para la inclusión de números dentro de los textos de las labels se necesita usar la función QString::number.

```
Asociadocon[i]->setText("Mota"+ QString::number(asociado[i]));
```

Finalmente se asocian las funciones creadas anteriormente a gbox, que es un objeto de tipo QGridLayout, y este se define como el Layout del QGroupBox:

```
gbox[i]->addWidget(Funcion[i],0,0);
gbox[i]->addWidget(Imagen[i],0,1);
gbox[i]->addWidget(Asociado[i],1,0);
gbox[i]->addWidget(Asociadocon[i],1,1);
grupo[i]->setLayout(gbox[i]);
```

Con esto se consigue que todos los elementos que se han ido añadiendo al Layout de gbox finalmente pasen al QGroupBox grupo[i] en la disposición la que se ha indicado, esto es:

-La función arriba a la izquierda (siempre debajo del título).

-Imagen arriba derecha.

- El texto “asociado” abajo a la izquierda.

-Mota a la que está asociada abajo a la derecha.

CAPITULO 6

INSTALACIÓN Y PUESTA EN MARCHA

6.1 AÑADIR Y QUITAR MOTAS

Para la construcción de la red de sensores y actuadores se puede proceder de dos formas. La primera es un pequeño tutorial (ver figura 35) que se ha realizado dentro de la interfaz de usuario, que lleva paso a paso hasta la incorporación de una mota a la red.

Para usuarios avanzados, es posible dirigirse directamente al esquemático de control donde puede hacerse lo mismo pero sin necesidad de tutorial.

Los pasos para incluir una mota en la red están descritos visualmente en el tutorial y son los siguientes:

Figura 38. Paso 1/3 creación de la red.

Figura 39. Paso 2/3 creación de la red.

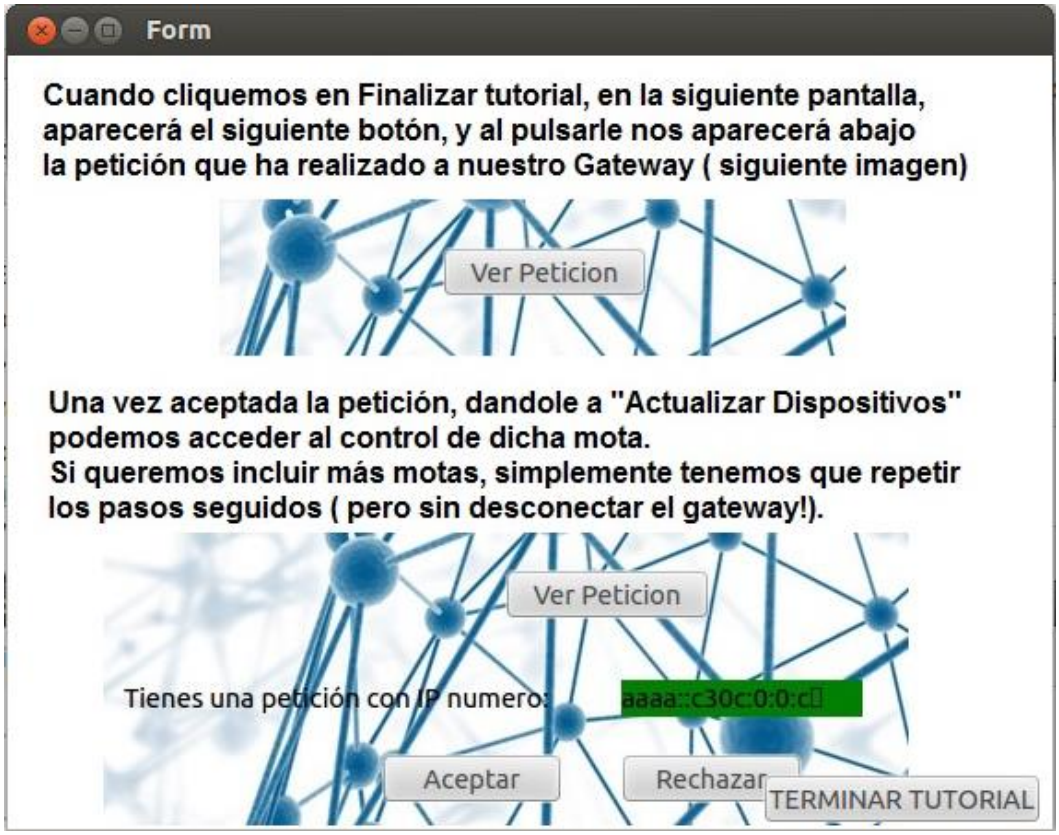


Figura 40. Paso 3/3 creación de la red.

Si se sigue todos estos pasos correctamente irá dando la opción de continuar con el tutorial. Si por ejemplo se está en el paso 1 y no se conecta el dispositivo al PC (o no tiene los permisos del puerto serie dados) no dejará continuar.

Una vez terminado el tutorial el programa redirigirá a la página del esquemático (a la que puede irse también directamente desde la pantalla inicial).

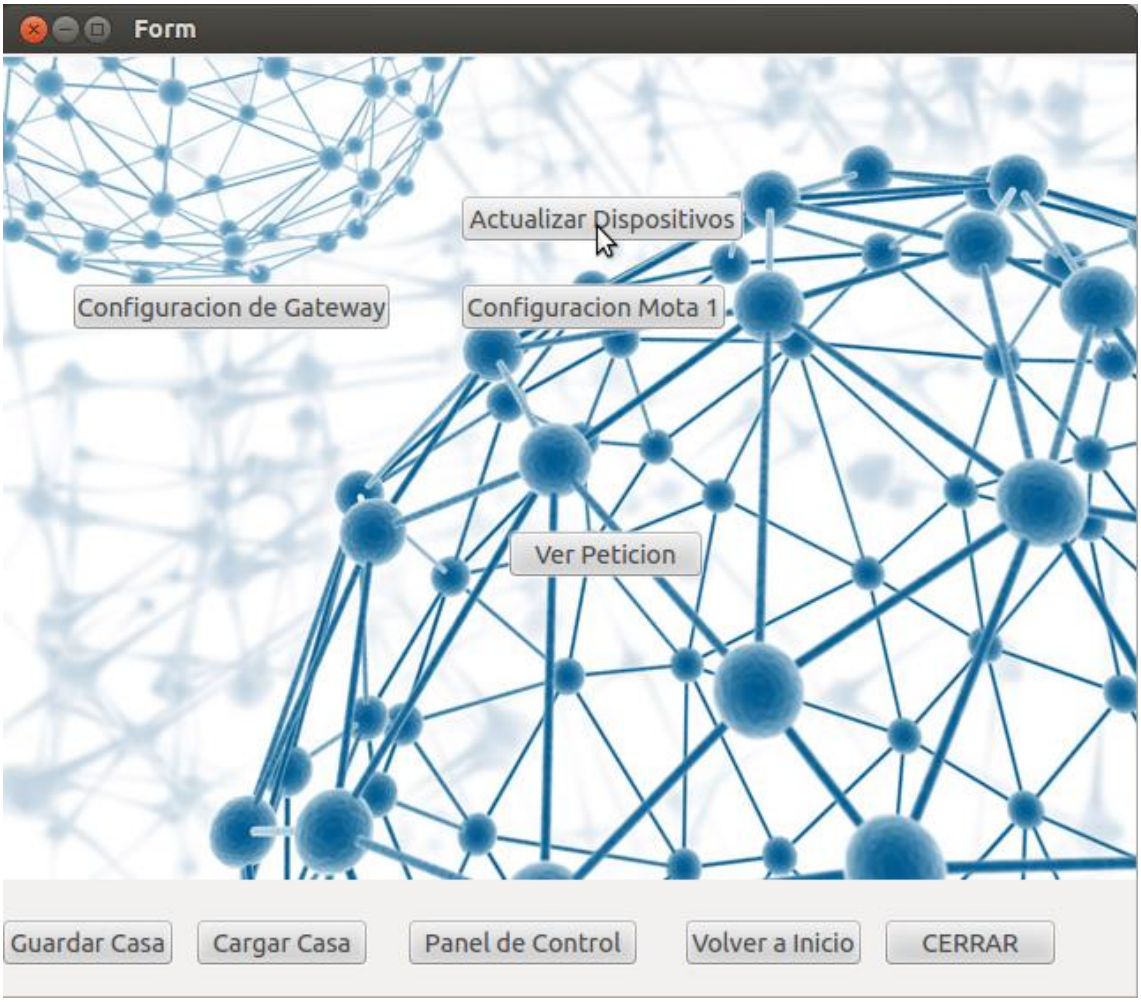


Figura 41. Si se sigue al pie de la letra el tutorial, llegaremos a la siguiente situación.

La mota inalámbrica número 1 ya ha sido añadida. Una vez en la página del esquemático (figura 39) pueden realizarse varias opciones, entre ellas las comentadas anteriormente: cargar y guardar casa. Para eliminar una mota de la red tan solo se tendría que ir a la página de configuración de dicha mota, en este caso Configuración Mota 1.

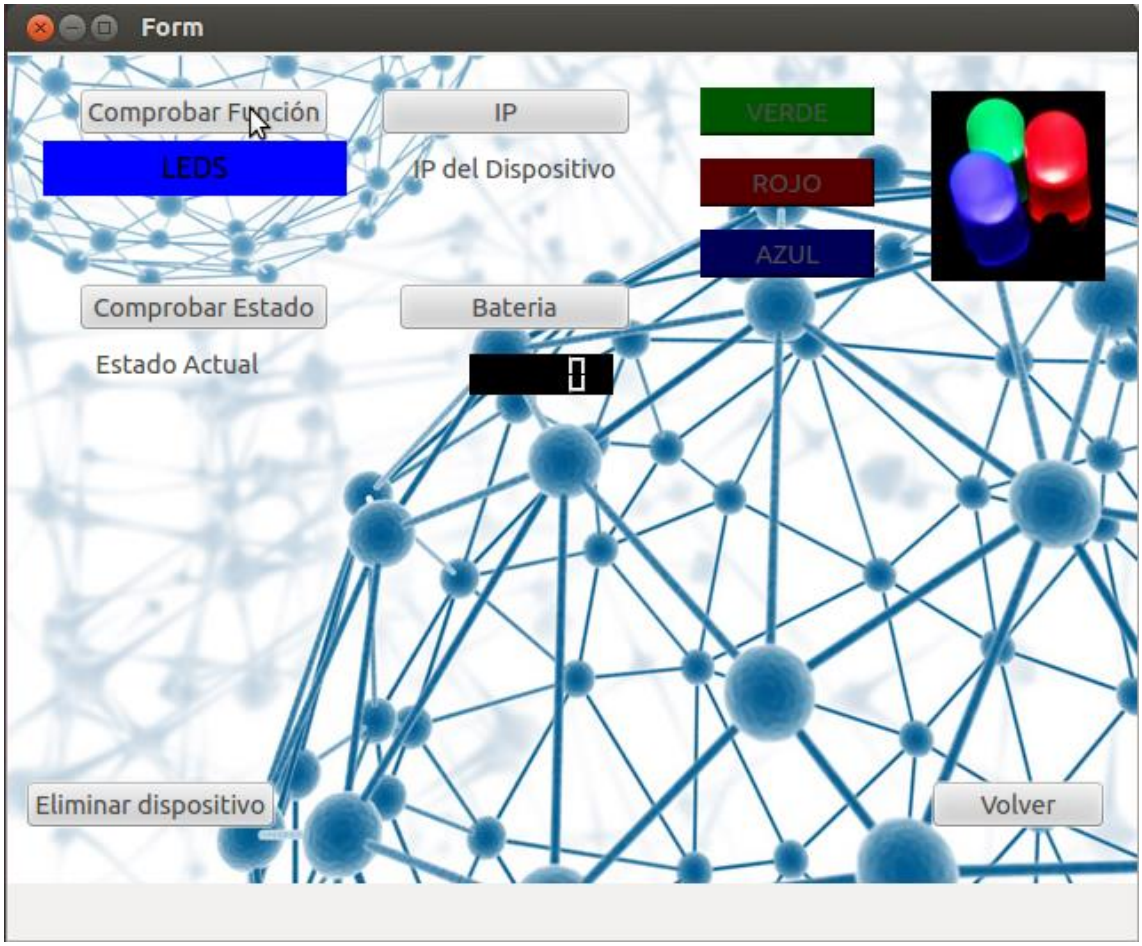


Figura 42. Pantalla de configuración de dispositivo, en este caso de LEDS.

Para borrar este dispositivo de la red, simplemente con darle a Eliminar Dispositivo quedaría totalmente borrado del sistema.

Esta acción enviaría un mensaje vía puerto serie al Gateway que haría lo propio con la dirección en la que está registrada la mota en cuestión.

6.2 ASOCIAR FUNCIONES

Algunas motas inalámbricas que hagan funciones concretas como un interruptor o un sensor de luz permitirán asociar dichas funciones a otros dispositivos para realizar acciones concretas. Por ejemplo asociar un interruptor, o el mismo Gateway al LED rojo de una mota con función LEDS. Esas asociaciones pueden hacerse desde la página de control de dispositivo:

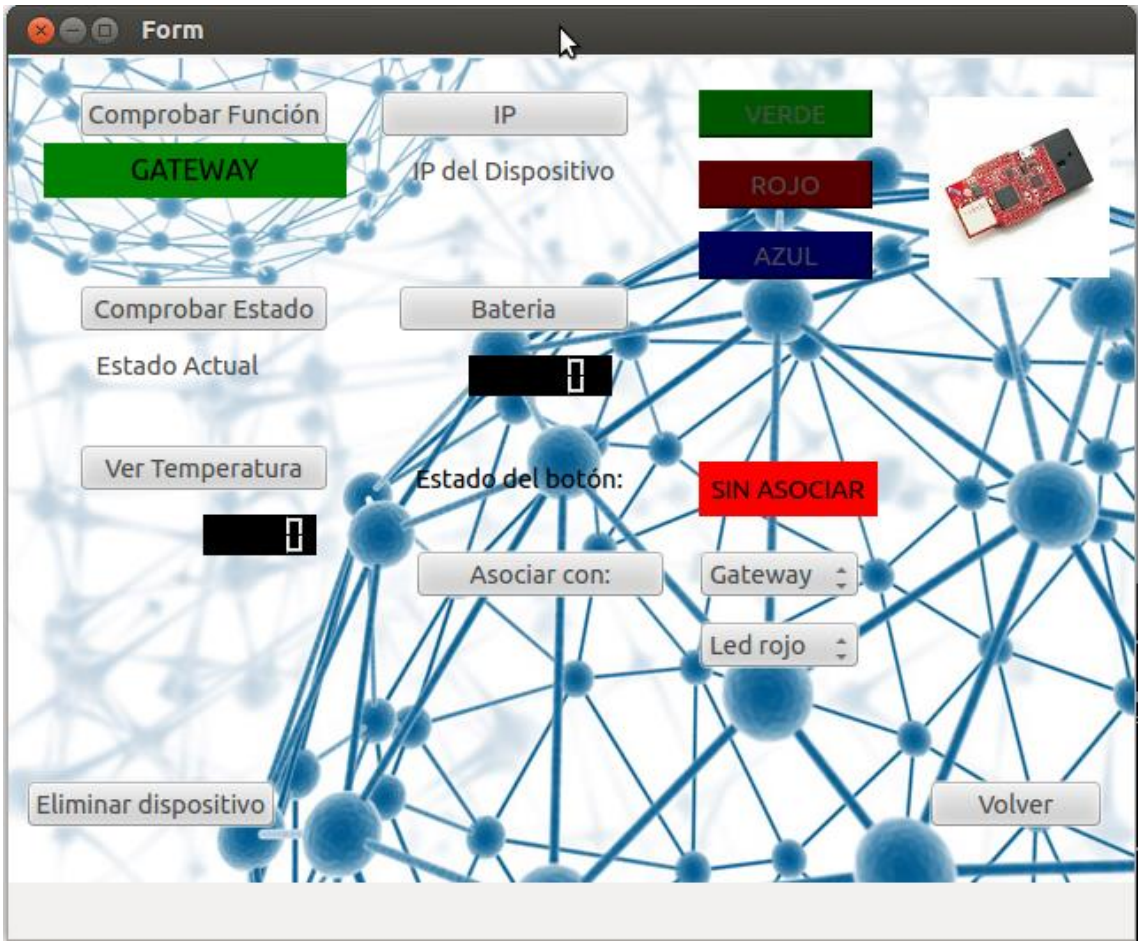


Figura 43. Pantalla de configuración de dispositivo, en este caso de Gateway.

En la parte central de la pantalla se indicará el estado de asociación del dispositivo.

En este caso el Gateway no tiene su botón asociado a ninguna mota, pero abriendo los dos desplegables daría la opción de asociarlo con los dispositivos que así los permitan sus funciones.

No solo se permite asociar botones sino también sensores, con el fin de permitir también una cierta autonomía de funciones.

Se podría por ejemplo asociar un sensor de luz con una bombilla, por ejemplo.

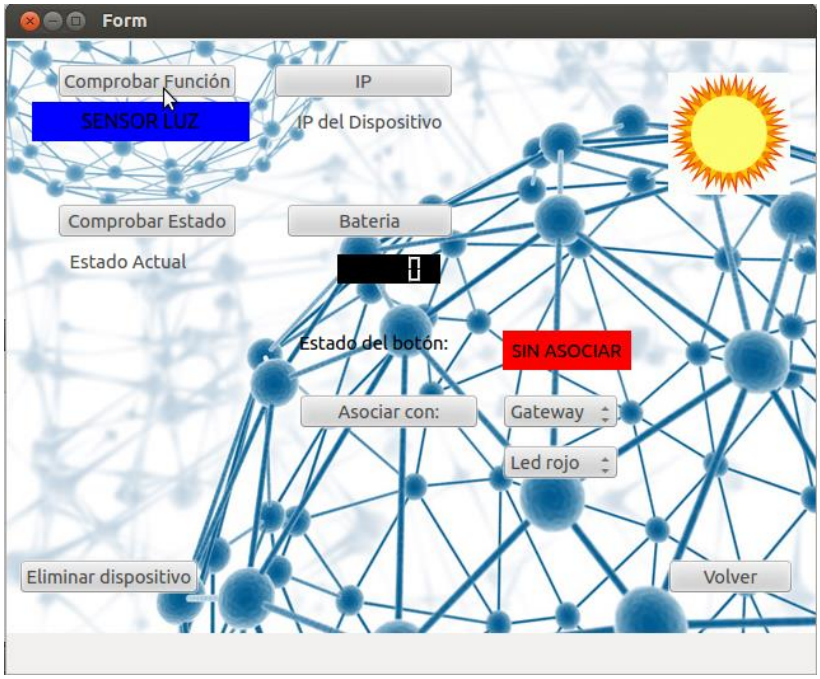


Figura 44. Pantalla de configuración de dispositivo, en este caso de un Sensor de Luz.

Así, asociando un sensor de luz a otro dispositivo, hará que se le envíen periódicamente datos a dicho dispositivo receptor, que será un actuador, para trabajar con esos datos como considere conveniente (si es muy bajo encender luces, si es muy alto apagarlas…). Si todo se ha realizado correctamente se verá del siguiente modo:

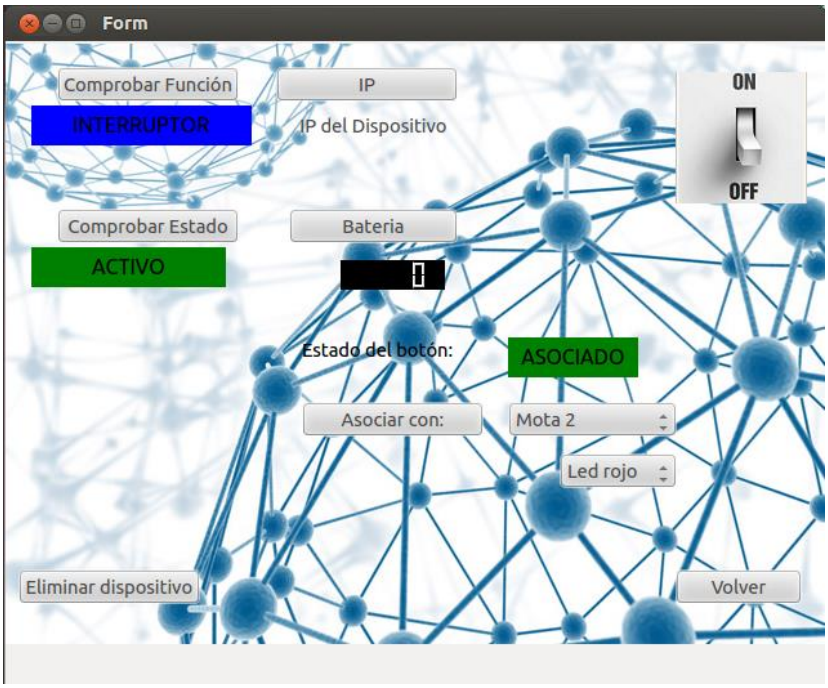


Figura 45. Asociación realizada correctamente.

6.3 MOTAS VIRTUALES Y PANEL DE CONTROL

A pesar de todos los programas mono función que se han programado, el material existente (2 motas inalámbricas y 1 mota Gateway) no permiten una expansión mayor del programa. Para hacer una visualización y una posible simulación de cómo quedaría el programa con una red de sensores algo mayor, se han creado en el programa varias motas virtuales que simulan un calefactor, un sensor de presencia y una puerta (que se abre con el anterior sensor de presencia).

Estas motas pueden ser incluidas en el programa una vez se han añadido las dos motas inalámbricas iniciales y aparecerán de forma simultánea las tres en el esquemático de redes:

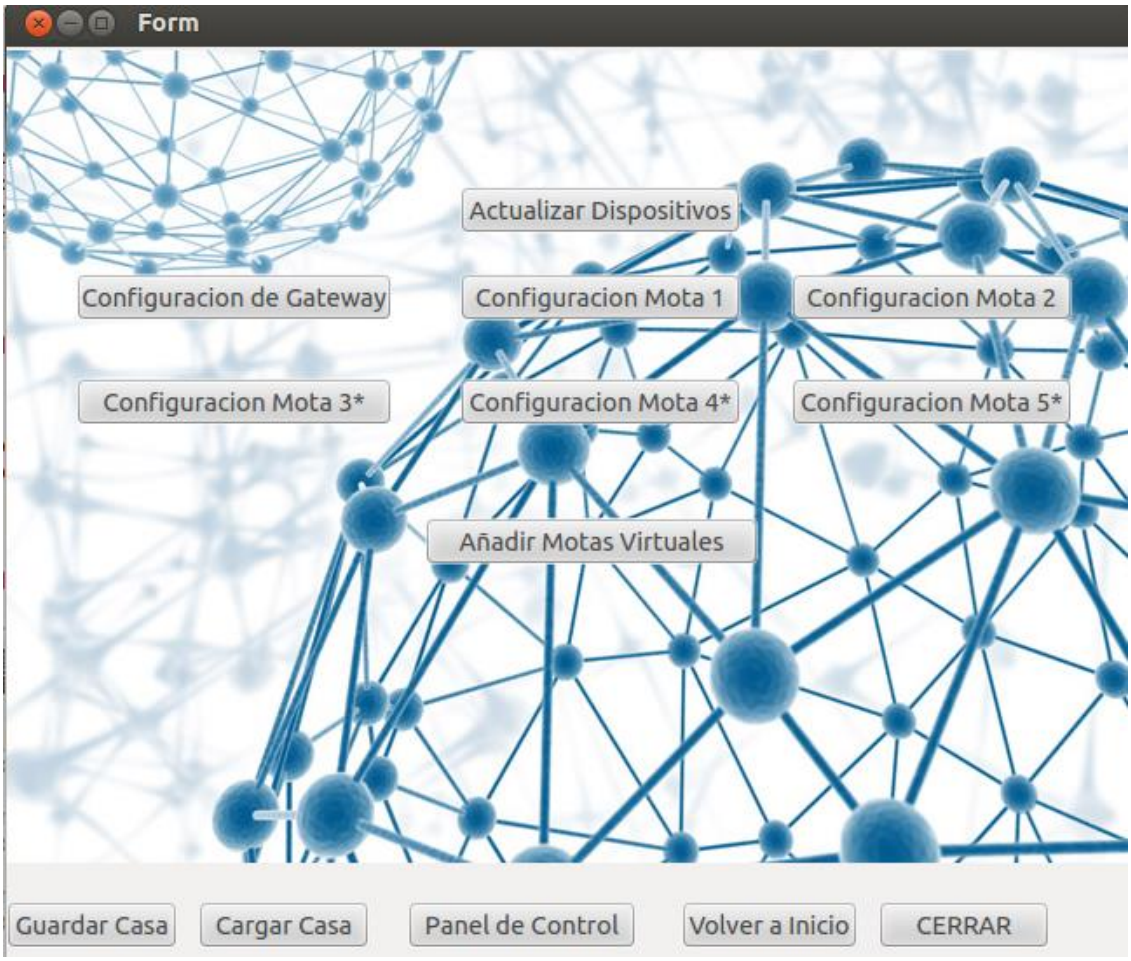


Figura 46. Vista del esquemático con las Motas Virtuales añadidas.

Las motas virtuales van a tener todos los valores prefijados y no pueden ser asociadas a ninguna mota real. También se puede acceder a su hoja de control de dispositivo, aunque realmente no se realiza ninguna acción:

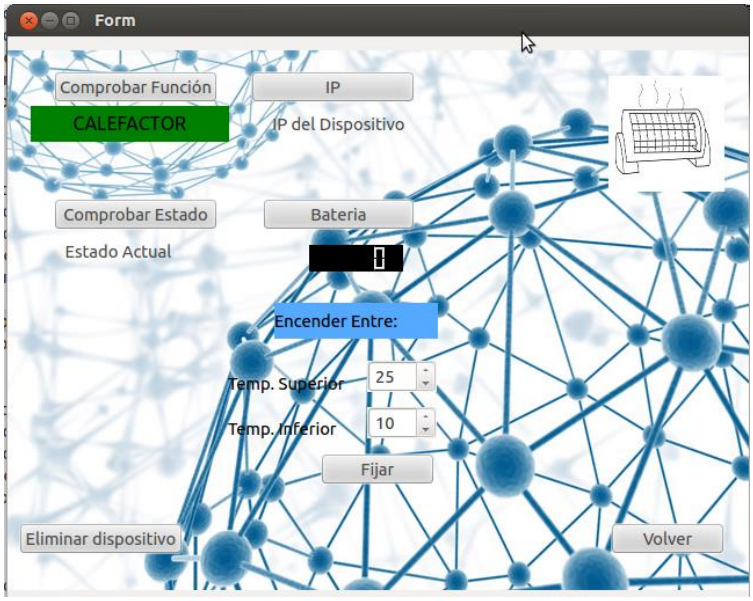


Figura 47. Configuración de un dispositivo virtual, calefactor.

Para tener una visualización no solo mota por mota sino del conjunto de la red que se esté creando, existe también la opción de entrar en el panel de control, accediendo desde el esquemático de conexiones (Por ejemplo, Figura 463) en el panel se irán viendo las motas según se han incluido a la red, con el detalle de su función y visualizaremos las asociaciones entre motas.

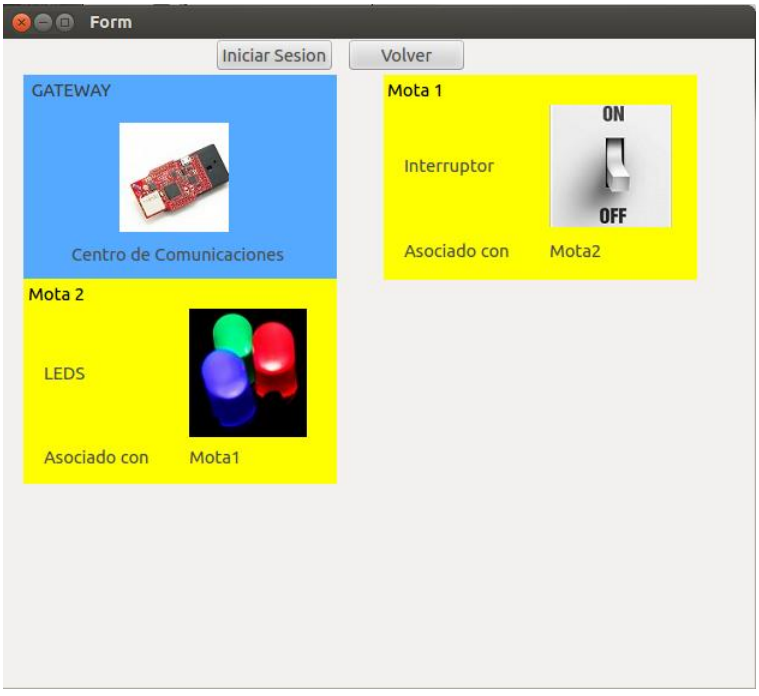


Figura 48. Panel de control con el Gateway y dos motas asociadas entre sí.

Una vez hayamos añadido las motas virtuales, estas también aparecerán en el panel de control. Incluso si hemos hecho algún tipo de vínculo entre ellas, a pesar de ser completamente virtual, también aparecerá en la pantalla:

Figura 49. Panel de Control con Gateway+ 2 Motas inalámbricas+3 Motas virtuales.

En este caso se ha asociado la mota virtual 4 a la mota virtual 5, que corresponden a un sensor de presencia y a una puerta automática respectivamente.

La visualización de todos los sensores en este caso es la misma debido a que son o actuadores (Motas 2, 3 y 5) o sensores de tipo booleano, ya que tanto el interruptor (Mota 1) como el sensor de presencia (Mota 4) o están activados o no.

En el caso de sensores de tipo numérico como Temperatura o Luz, en vez de “Asociado con” se incluiría un “Midiendo” y posteriormente la medida del sensor. La siguiente figura es el ejemplo de lo anterior:

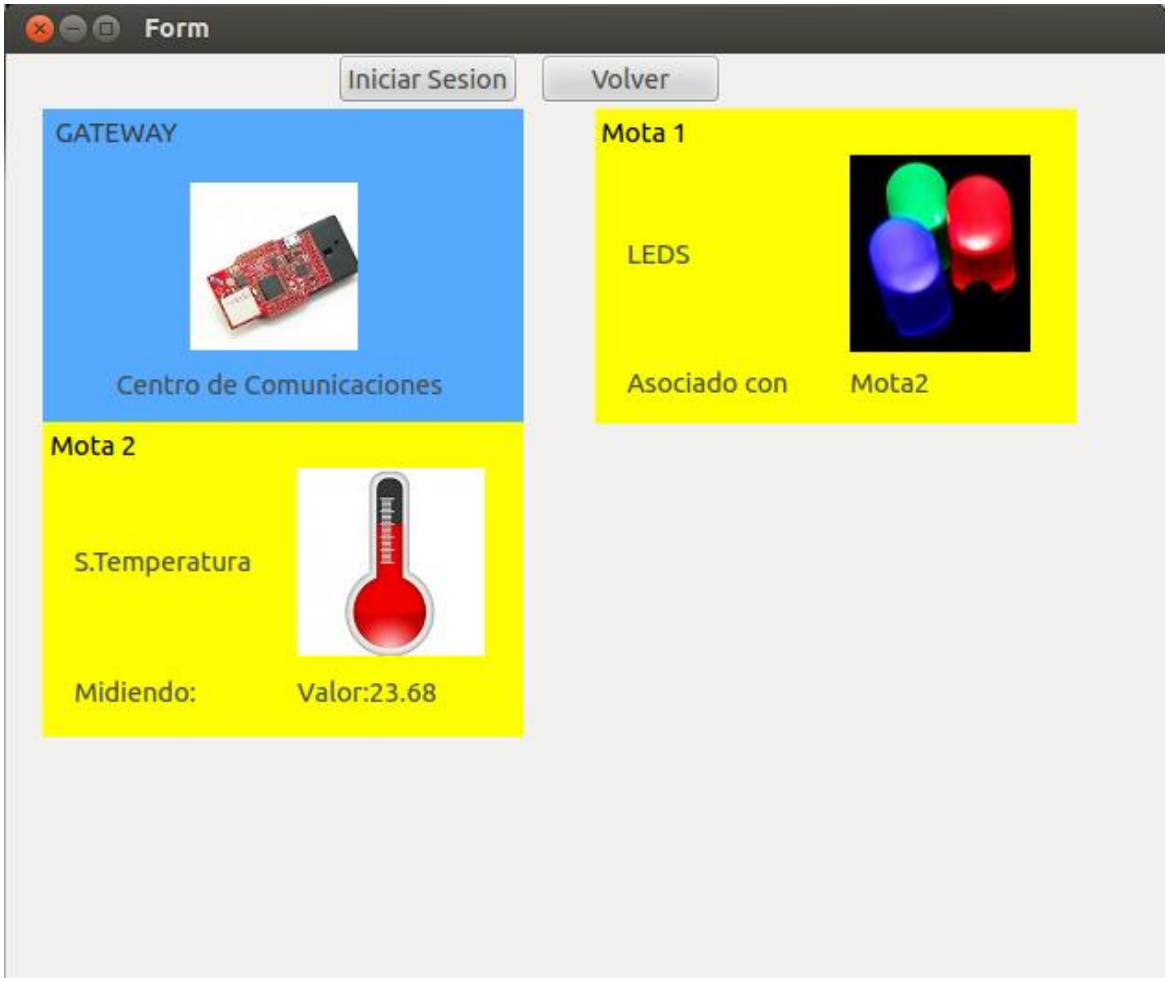


Figura 50. Panel de Control con Gateway+ 2 Motas inalámbricas, una de ellas sensor.

La asociación de los LEDS en este caso con el sensor de temperatura ha sido programada para que en diversos rangos de temperatura se enciendan unos u otros LEDS. Estos valores actualmente son modificables desde el programa de temperatura y aun no se ha implantado para su modificación en la interfaz de control, quedando como trabajo futuro.

El sensor de luminosidad no ha sido habilitado para este tipo de medición ya que el utilizado tiene un comportamiento algo inestable, no siempre funcionando con corrección y haciendo reiniciarse cada cierto tiempo a la mota.

El botón inicio sesión es un posible proyecto futuro para el reseteo automático cada X tiempo de los valores que aparecen en el panel de control, algo que a la conclusión de este proyecto solo puede realizarse manualmente.

CAPITULO 7

GESTIÓN DEL PROYECTO

7.1 DIAGRAMA DE GANTT

Figura 51. Diagrama de Gantt del Trabajo Fin de Grado.

7.1 PRESUPUESTO

El presupuesto del proyecto se ha realizado mediante la suma de los costes laborales y los costes materiales del mismo. Hay que tener en cuenta que una gran parte del proyecto se ha realizado con software gratuito (tanto Qt Creator como Contiki o Linux lo son) y por eso los costes de material son bastante reducidos.

No se ha incluido el coste del PC usado para el proyecto ni cables USB adicionales para pruebas.

COSTE DEL MATERIAL			
CONCEPTO	C / U	CANTIDAD	TOTAL
Pack Zolertia Z1 Platform	95	3	285
Mota Z1	Incluido	3	0
Par Baterias AA	Incluido	3	0
Sujección Baterias	Incluido	3	0
Phidgets & Ziglet	5	2	10
Cable USB	7,5	1	7,5
ZIG002 Light Sensor	29,95	1	29,95
Linux Ubuntu	Gratuito	1	0
Qt Creator	Gratuito	1	0
PuTTY	Gratuito	1	0
Dia	Gratuito	1	0
Microsoft Office Starter	Gratuito	1	0
SO Contiki	Gratuito	1	0
VMWare Player	Gratuito	1	0
		TOTAL	**322,45**

Tabla 5. Costes Materiales.

Para el coste laboral se ha realizado una estimación de las horas dedicadas en los diferentes periodos que se ha trabajado en el proyecto a fin de determinar el número de horas totales empleadas y ver así el coste final.

COSTE LABORAL				
CONCEPTO	HORAS SEMANALES	CANTIDAD TOTAL HORAS	C/U	TOTAL
Horas de Tutoria				
Horas ingeniero jefe	20	20	50	1000
Horas ingeniero	20	20	30	600
Horas Periodo Febrero-Junio 2013	5	100	30	3000
Horas Periodo Julio-Septiembre 2013	25	300	30	9000
Horas Periodo Octumbre-Diciembre 2013	4	48	30	1440
Horas Periodo Enero-Febrero 2014	50	400	30	12000
				TOTAL 27040

Tabla 6. Costes Laborales

Así pues, el los costes totales del proyecto son los siguientes:

COSTES TOTALES	
CONCEPTO	PRECIO
Costes Materiales	322,45
Costes Laborales	27040
TOTAL	27362,45

Tabla 7. Costes totales.

El presupuesto total de este proyecto asciende a la cantidad de VEINTISIETE MIL TRESCIENTOS SESENTA Y DOS EUROS CON CUARENTA Y CINCO CENTIMOS .

Leganés a X de Septiembre de 2013

Guillermo Cañada Valverde.

CAPITULO 8

CONCLUSIONES Y TRABAJOS FUTUROS

La realización de este proyecto abre un abanico de posibilidades muy amplias de desarrollo por varios frentes.

En primer lugar sería interesante el desarrollo de motas inalámbricas de bajo coste que actúen como sensores o actuadores. Actualmente los sensores más baratos que incorporen una antena se sitúan en torno a los 50 euros y es un precio prohibitivo para la expansión de este mercado. El desarrollo de los protocolos inalámbricos de baja potencia necesita a su vez de un empujón tecnológico para abaratar el precio de las antenas, que es lo que más encarece los sensores en la actualidad.

Por otro lado, la entrada de los dispositivos móviles, tablets y demás en el mercado abre la posibilidad de no solo poder administrar tu casa desde un PC sino con aplicaciones para móviles o dispositivos más rápidos y cómodos de usar.

A su vez, con esas esas innovaciones, el manejo remoto de las mismas, es decir, con un servidor en internet desde el que puedas controlar tu casa, sería una vía de investigación muy interesante también.

Manejar dispositivos de tu casa y tener todo controlado mediante una aplicación por ejemplo de Android es una realidad actualmente.

Trasladar todo eso a un sistema de sensores y actuadores inalámbricos que sean de bajo coste y además con la capacidad de plug-and-play, sin requerir conocimientos técnicos, es un futuro bastante cercano.

A pesar de haber realizado este proyecto con Zolertia Z1 y gustarme mucho por las facilidades y capacidades técnicas de las motas, me decanto más por que tales avances puedan venir más de la mano de Arduino, por su filosofía de Software y Hardware libre que tan de moda está últimamente y que confía en el desarrollo de sus productos más al hobby de miles y miles de aficionados a la informática en general y a la domótica en particular, que de solo una empresa que “hace la guerra por su cuenta”.

Mi conclusión es que siendo un campo, el Internet of Things, con unas posibilidades de futuro enormes creo que aún necesita de tiempo, desarrollo y esfuerzos antes de ser algo que se expanda al gran público.

Además, añadir que no sé si solo pasará con Zolertía o con el resto de motas, la comunicación inalámbrica de este tipo aún tiene que perfeccionarse para evitar pérdidas de paquetes y multitud de “cosas raras” que hacen las motas.

Me gustaría destacar que más allá de toda esta "presentación de resultados" que es la memoria, detrás hay muchísimo trabajo, decenas de programas de prueba, un programa entero, muy extenso, de desarrollo de interfaz gráfica además del que he presentado en esta memoria. En general he tenido muchos momentos buenos pero también muchos malos, con muchas horas y pocos avances en algunos momentos.

Para mí como estudiante lo que me ha aportado este proyecto ha sido muchísima información en este campo, aprender un metodología de trabajo y exigirme una gran dedicación, que por supuesto, solo le dedicas a las cosas que te gustan, aunque a veces vengan fracasos.

En general creo que todo lo que he aprendido ha merecido mucho la pena, que es un campo en el que me gustaría profundizar laboralmente hablando y seguir con ello como hobby.

En resumen, estoy contento con el trabajo realizado, con el esfuerzo que el tutor ha puesto en mí y a pesar de ser un campo todavía muy en desarrollo y cuya investigación puede resultar tanto en un día brillante con mucho avance como una semana perdida por una "vía muerta", el cómputo general es muy positivo.

GLOSARIO:

WSN -> Wireless Networking.

PC -> Personal Computer.

OS -> Operative System. (SO : Sistema operativo)

WPAN -> Wireless Personal Area Network.

IEEE -> Institute of Electrical and Electronics Engineers.

OSI -> Open System Interconnection.

6LoWPAN -> IPv6 over Low Power Wireless Personal Area Network.

MAC -> Media Access Control.

LLC -> Logical Link Control.

IDE -> Integrated Development Environment.

ID -> Identification number.

REFERENCIAS:

WEBS

-[1] Información sobre 802.15.1. Wikipedia (última edición 2014). Bluetooth. Consultada el 12 de febrero de 2014 en: http://es.wikipedia.org/wiki/Bluetooth

-[2] Información sobre 802.15 en general. Wikipedia (última edición 2013). IEEE 802.15. Consultada el 12 de febrero de 2014 en: http://es.wikipedia.org/wiki/IEEE_802.15

-[3] Información sobre Zigbee. Wikipedia (última edición 2013). Zigbee (especificación). Consultada el 12 de febrero de 2014 en: http://es.wikipedia.org/wiki/ZigBee_(especificaci%C3%B3n)

-[4] Información sobre 6LoWPAN. Wikipedia (última edición 2014). 6LoWPAN (versión en inglés). Consultada el 12 de febrero de 2014 en: http://en.wikipedia.org/wiki/6LoWPAN

-[5] Información e imágenes sobre motas inalámbricas MICAz y TelosB. Memsic (2010). Wireless Modules. Consultada el 12 de febrero de 2014 en: http://www.memsic.com.php5-12.dfw1-1.websitetestlink.com/products/wireless-sensor-networks/wireless-modules.html

-[6] Información e imágenes sobre motas inalámbricas WaspMote de Libelium. Libelium (*). Consultada el 18 de febrero de 2013 en: http://www.libelium.com/es/products/waspmote/

- [7] Información e imágenes sobre las motas Arduino y diferentes componentes. Arduino(2014). Consultada el 18 de febrero de 2013 en: http://www.arduino.cc/es/

-[8]Datasheet del Zolertia Z1. Zolertia(*). Z1 Platform Datasheet. Consultada el 12 de febrero de 2014 en: http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf

-[9] Datasheet del ZIG002. Zolertia (*). Brochure Digital Ambient Light Sensor (ZIG002). Consultada el 12 de febrero de 2014 en: http://zolertia.sourceforge.net/wiki/images/3/39/Brochure_Zig002.pdf

-[10] Información sobre QtCreator. Wikipedia (última edición 2013). Qt Creator. Consultada el 14 de febrero de 2014 en: http://en.wikipedia.org/wiki/Qt_Creator

-[11] Guía de inicio con Contiki/Cooja. Contiki-OS(*). Get Started with Contiki. Consultada el 12 de febrero de 2014 en: http://www.contiki-os.org/start.html

-[12] Guía de burneo de ID. Sourceforge Zolertia(2013). Mainpage: Contiki Apps. Consultada el 15 de febrero de 2014 en: http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_apps

INFORMACIÓN RELEVANTE:

Los programas comentador y resumidos anteriormente están basados en programas creados y distribuidos por Swedish Institute of Computer Science, y en concreto por Adam Dunkels. Su uso total o parcial exige ir acompañado de este pequeño fragmento de texto. Con él, aparte de cumplir la legalidad, quiero agradecer el trabajo previo de todas las personas que se ha usado y en el que me he apoyado para la realización de este proyecto

```
/*
* Copyright (c) 2011, Swedish Institute of Computer Science.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. Neither the name of the Institute nor the names of its contributors
*    may be used to endorse or promote products derived from this software
*    without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED.  IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* This file is part of the Contiki operating system.
*
*/
```